4

Applicability of JEE Technology

CERTIFICATION OBJECTIVES

Thin-client multi-tiered applications are difficult to develop because they involve many lines of intricate program code and configuration files to handle transaction and state management, multithreading, resource pooling, and other complex low-level details. The component-based and platform-independent JEE architecture facilitates such development because business logic is organized into reusable components. Moreover, the JEE server provides underlying services in the form of a container for every component type. In this chapter, we will explore this architecture in detail and discuss when and why JEE is a better architectural choice.

CERTIFICATION OBJECTIVE 4.01

Explain the JEE Architecture and System Requirements

The Java Enterprise Edition (JEE) platform uses a multi-tiered distributed application model, in which application logic is divided into components according to function. The various components that a JEE application comprises are installed on different machines. A component's location depends on which tier or layer that component belongs to in the multi-tiered JEE environment. Figure 4-1 shows two multi-tiered JEE applications divided into the tiers described here:

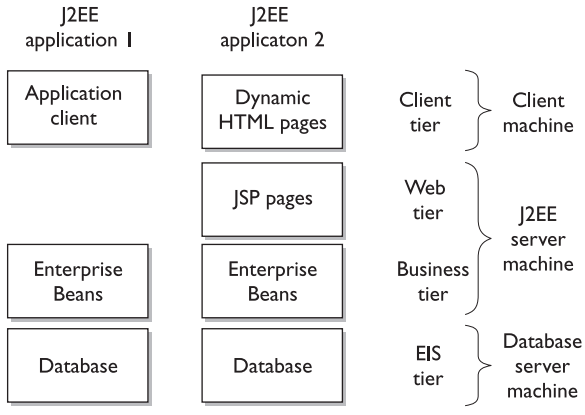
- Client tier components run on the client machine.
- Web tier components run on the JEE server.
- Business tier components run on the JEE server.
- Enterprise Information System (EIS) tier software runs on the EIS server.

JEE Technology Layers Applied

While a JEE application can consist of three or more tiers, JEE multi-tiered applications are generally considered to be three-tiered applications, because they are distributed across three different locations: client machines, the JEE server machine, and the database or legacy machines at the back end. Three-tiered applications that run in this way extend the standard two-tiered client/server model by placing a multithreaded application server between the client application and back-end storage.

FIGURE 4-1

Multi-tiered JEE applications



The Client Layer

The client layer or tier of a web application is typically implemented as Hypertext Markup Language (HTML) displayed in a web browser running on the user's machine. It may also be implemented using Java. Its function is to display data, providing the user with a place to enter and update data. Generally, one of two common approaches is used for building the client layer:

- **A pure HTML-only client** In this scenario, virtually all of the intelligence is placed in the middle tier. When the user submits the web pages, all the validation is done on the JEE server (the middle tier). Errors are then posted back to the client.
- **A hybrid HTML/Asynchronous JavaScript and XML (AJAX)/JavaScript client** In this scenario, some intelligence is included in the web pages, which run on the client. The client will do some basic validations (for example, ensuring that mandatory columns are completed before allowing users to submit information). The client may also include some AJAX for functions such as hiding fields when they are no longer applicable due to earlier selections.

The pure HTML approach is less efficient for end users because all operations require the server for even the most basic functions. On the other hand, as long as the browser understands HTML, it will generally work with this basic approach, making it possible to work on basic wireless or text-only browsers. The second

argument in favor of this approach is that it provides a better separation of business logic and presentation. The hybrid client approach is more user-friendly, requiring fewer trips to the server. Typically, AJAX and JavaScript are written to work with more recent versions of mainstream browsers.

As mentioned, the JEE application client may also provide client layer functionality, providing the user with a place to maintain data. A JEE application client is a thick (RMI-IIOP [Remote Method Invocation–Internet Inter-ORB Protocol]) Java application; it differs from a stand-alone Java application client in that it is a JEE component. Like other JEE components, a JEE application client is created with the application deployment tool and added to a JEE application.

Because it is part of a JEE application, a JEE application client has two advantages over a stand-alone Java application client: First, a JEE application client is bundled in an Enterprise Archive (EAR) file with all the required software making it easily portable—that is, it will run on any JEE-compliant server. Second, because it is bundled with the required libraries, it has access to the full array of JEE services. Its weight sometimes makes it a poor choice, and the emergence of the JavaServer Page (JSP) has curbed its popularity.

The Presentation Layer

The presentation layer generates web pages and any dynamic content in the web pages. The dynamic content is typically obtained from a database; for example, content may consist of a list of transactions conducted during the last month. The other major job of the presentation layer is to package requests contained on the web pages coming back from the client.

The presentation layer can be built with a number of different tools. The presentation layers for the first web sites were built as Common Gateway Interface (CGI) programs. Netscape servers also offered server-side JavaScript for web sites. Contemporary web sites generally have presentation layers built using the Microsoft solution, Active Server Pages (ASP), which may be generated by Visual InterDev, or they use the Java solution, which utilizes some combination of servlets and JSP. Tools provide methods to facilitate embedding dynamic content inside other static HTML in the web page. They also provide tools for simple parsing of the web pages coming back from the client to extract the user-entered information.

The presentation layer is generally implemented inside a web server (such as Microsoft IIS, BEA WebLogic, or IBM WebSphere). The web server typically handles requests for several applications in addition to requests for the site's static web pages. Based on the initial configuration, the web server knows to which application to forward the client-based request (or which static web page to serve up).

The Business Logic Layer

The bulk of the application logic is written in the business logic layer. The challenge here is to allocate adequate time and resources to identify and implement this logic. Business logic includes the following:

- Performance of all required calculations and validations
- Workflow management (including keeping track of session data)
- Management of all data access for the presentation layer

In modern web applications, business logic is frequently built using the Java solution, with Enterprise JavaBeans (EJB) that are built to carry out the business operations. Language-independent Common Object Request Broker Architecture (CORBA) objects can also be built and accessed with a Java presentation tier. The main component of CORBA is the Object Request Broker (ORB). It encapsulates the communication infrastructure necessary to locate objects, manage connections, and deliver data. The ORB core is responsible for the communication of requests. The basic functionality provided by the ORB consists of passing the requests from clients to the object implementations on which they are invoked. The ORB then transfers the request to the object implementation, which receives the request, processes it, and returns an object result.

Much like the presentation layer, the business logic layer is generally implemented inside the application server. The application server automates many services, such as transactions, security, persistence/connection pooling, messaging, and name services. Isolating the business logic from the need to manage resources allows the developer to focus on building application logic. In the application server marketplace, vendors differentiate their products based on manageability, security, reliability, scalability, and tools support.

The Data Layer

The data layer is responsible for data management. A data layer may be as simple as a modern relational database; on the other hand, it may include data access procedures to other data sources such as nonrelational databases, legacy files, or message-oriented middleware. The data layer provides the business logic layer with required data when needed and stores data when requested.

To avoid making an application less interoperable, the architect should strive to keep validation and business logic out of the data layer; that logic belongs in the business logic layer. Sometimes basic database design rules can overlap with business logic. There is usually some basic business logic in the data tier. For example,

both *not null* constraints and *foreign key* constraints, which designate that certain columns must have a value and that the value must match an existing foreign row's corresponding key value, could be considered "business rules" that should be known only to the business logic layer. Most product designers would agree that it is necessary to include such simple constraints in the database to maintain data integrity, changing them as the business rules evolve.

JEE Application Components

JEE applications are made up of *components*: self-contained functional software units assembled into JEE applications with their related classes and files. These components communicate with other components. The JEE specification defines the following components:

- **Client components** Application clients and applets
- **Web components** Java Servlet and JSP technology
- **Business components** EJB components

These components are written in the Java programming language and compiled in the same manner as any other program written in Java. When working with the JEE platform, the difference is that JEE components are assembled into a JEE application, where it is verified that they are well formed and compliant with the JEE specification. They are then deployed to production, where they are run and managed by the JEE server.

Client Components

A JEE application can either be web-based or non-web-based. Non-web-based components are an extension of the heretofore common client/server applications. In a non-web-based JEE application, an application client executes on the client machine. For a web-based JEE application, the web browser downloads web pages and applets to the client machine.

Application Clients Application clients run on a client machine, providing a way for users to handle tasks such as JEE system or application administration. Usually, a graphical user interface (GUI) is created using Swing APIs; however, a command-line interface is also possible. Application clients directly access enterprise beans that run in the business tier. On the other hand, an application client can open an HTTP

connection establishing communication with a servlet running in the web tier if warranted by the JEE application.

Web Browsers The user's web browser downloads static or dynamic HTML, Wireless Markup Language (WML), eXtensible Markup Language (XML), or pages in other formats from the web tier. Servlets and JSPs running in the web tier provide the ability to generate dynamic web pages.

Applets Web pages downloaded from the web tier can include embedded applets. These are small client applications, written in the Java programming language, which execute in the Java Virtual Machine (JVM) installed in the web browser. Client systems often need an additional Java plug-in and perhaps even a security policy file so that the applet can successfully execute in the web browser.

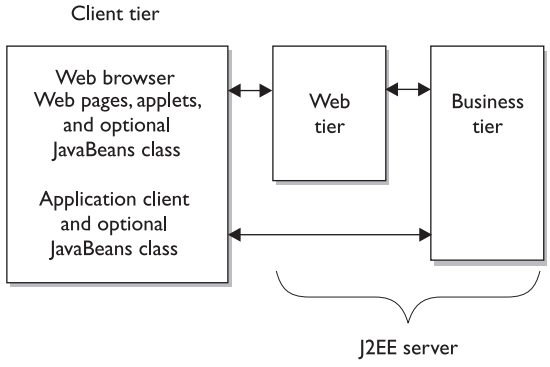
JSPs are the preferred API for the creation of web-based client programs, where plug-ins and security policy files are not necessary on the client system. In addition, JSPs enable cleaner, more modular application designs because they provide a way to separate application programming from web-page design. This means web-page designers do not need to know Java to do their jobs.

Applets running in other network-based systems such as handheld devices and cell phones are able to render WML pages generated by a JSP or servlets running on the JEE server. The WML page is delivered using the Wireless Application Protocol (WAP). The network configuration requires a gateway to translate WAP to HTTP and back again. This gateway translates the WAP request from the handheld device to an HTTP request for the JEE server, translating the HTTP server response and WML page to a WAP server response and WML page for display on the device.

JavaBeans Component Architecture The client tier sometimes includes a component based on the JavaBeans component architecture for managing data flow between the application client or applet and components running on the JEE server. The JEE specification does not regard JavaBeans components as components. As will be explained later in this book, JavaBeans are not the same as EJBs. JavaBeans components have instance variables as well as get and set methods for accessing the data in those instance variables. When used in this manner—that is, as a place to persist user entered data—JavaBeans components tend to be simple in design and implementation. They should, however, conform to the naming and design conventions specified in the JavaBeans component architecture.

FIGURE 4-2

Elements of the client tier



JEE Server Communications Figure 4-2 shows the various elements that make up the client tier. The client communicates with the business tier running on the JEE server either directly (as in the case of a client running in a browser) or by going through JSPs or servlets running in the web tier.

Thin Clients JEE applications use a lightweight interface to the application, known as a *thin client*, which does not perform functions such as querying databases, executing complex business rules, or connecting to legacy applications. Instead, these operations are off-loaded to web or enterprise beans that execute on the JEE server. Here, the security, speed, services, and reliability of JEE server-side technologies are maximized.

Web Components

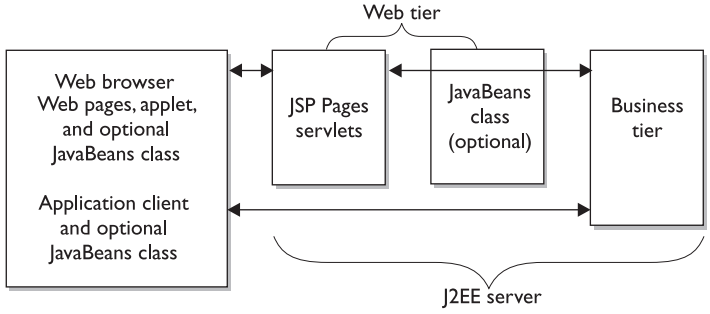
JEE web components are either JSP pages or servlets. Servlets are Java classes that dynamically process requests and construct responses. JSP pages are text-based documents containing static content along with snippets of Java code used to generate dynamic content. When a JSP page loads, a background servlet executes the code snippets, returning a response.

Although static HTML pages and applets are bundled with web components during application assembly, they are not considered web components by the JEE specification. In the same manner, server-side utility classes are often bundled with web components yet are not themselves considered web components.

The web tier, shown in Figure 4-3, might include JavaBeans objects for managing user input, sending that input to enterprise beans running in the business tier to be processed. JavaBeans that encapsulate UI controls or other dynamic functionality can be used to emulate the client/server user interface on a web-based page.

FIGURE 4-3

Elements of the web tier



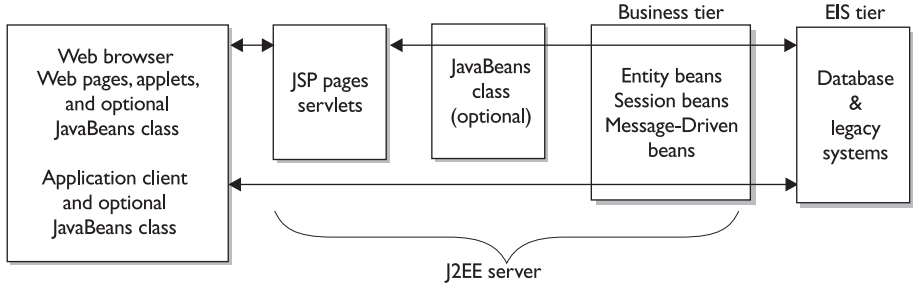
Business Components

Business code is logic that solves the functional requirements of a particular business domain such as banking, retail, or finance. This code is handled by enterprise beans that run in the business tier. Figure 4-4 demonstrates how an enterprise bean receives data from client programs, processes it, and then sends it to the enterprise information system tier to be stored. In addition, an enterprise bean retrieves data from storage, processes it, and then sends it back to the client program.

There are three kinds of enterprise beans: session beans, entity beans, and message-driven beans. Session beans represent transient conversations with a client. When the client completes execution, the session bean and its accompanying data are gone. On the other hand, entity beans represent persistent data, which typically is stored in one row of a database table. Entity beans can map to more than one row/record of a relational database table or legacy data store. If the client quits or the server shuts down, underlying services ensure that the entity bean data is saved. Message-driven beans feature a session bean and a Java Message Service (JMS) message listener. They allow business components to receive asynchronous JMS messages.

FIGURE 4-4

Business and EIS tiers



Enterprise Information System Tier The EIS tier is a giant "catch-all" for handling EIS software. It includes enterprise infrastructure systems such as Enterprise Resource Planning (ERP), mainframe transaction processing, database systems, and other legacy information systems. JEE application components access EISs for functions such as database connectivity. The reality is that most enterprise computing environments typically have a legacy system to maintain the so-called "books and records" of the firm. Moreover, new applications based on JEE will have to be able to interface and communicate request and response processing with the legacy systems. Architects must be able to build and use EIS to handle this important requirement.

To this end, the JEE Connector Architecture (JCA) is a standard architecture for connecting to EIS from the JEE platform. The JCA architecture defines a set of scalable, secure, and transactional mechanisms that describe the integration of EISs to an application server and enterprise applications. This architecture enables an EIS vendor to provide a resource adapter for its EIS that can be plugged into any application server that supports the JEE Connector Architecture. IBM in particular has embraced this standard to open access to mainframe-based online transaction processing (OLTP) systems such as Customer Information Control Systems (CICS).

JEE Architecture

Typically, thin-client, multi-tiered applications are difficult to write because they involve complex programming for handling transaction management, multithreading, database connection pooling, and other low-level details. The component-based and platform-independent JEE architecture makes JEE applications desirable and easier to develop because business logic is organized into reusable components, and the JEE server provides underlying services in the form of a container for every component type.

Containers and Services

Components are installed in their containers during deployment. *Containers* are the interface between a component and the platform-specific functionality supporting that component. Before a web component can be executed, it must first be assembled into a JEE application and then deployed into its container.

The process of assembly involves specifying container settings for each component within the JEE application as well as for the application itself. These settings customize the underlying support provided by the JEE server, including services such as security,

transaction management, Java Naming and Directory Interface (JNDI) lookups, and remote connectivity. Following are some examples:

- The JEE security model allows configuration of a web component or enterprise bean so that only authorized users can access system resources.
- The JEE transaction model provides for relationships among methods that make up a single transaction; therefore, all methods in one transaction are treated as a single unit of work.
- JNDI lookup services provide an interface to multiple naming and directory services in the enterprise, e.g., LDAP, allowing application components to access naming and directory services.

The JEE remote connectivity model manages the communication between clients and enterprise beans. After an enterprise bean is created, methods are invoked on it by the client as if it were in the same virtual machine.

Because the JEE architecture provides configurable services, application components within the same JEE application can behave differently, depending on where they are deployed. For instance, an enterprise bean can have security settings, allowing it a certain level of access to database data in one production environment and a different level of database access in another production environment.

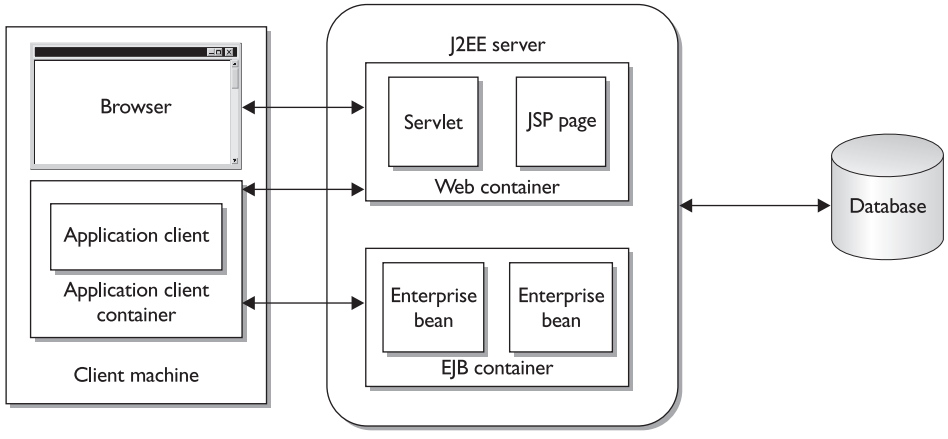
Containers also manage services such as enterprise bean and servlet life cycles, database connection resource pooling, data persistence, and access to the JEE platform APIs. Although data persistence is a nonconfigurable service, the JEE architecture allows you to include code in your enterprise bean implementation to override container-managed persistence (CMP) when more control is desired than the default provided by CMP. For example, bean-managed persistence (BMP) may be used to implement your own finder methods or to create a customized database cache.

Container Types

The deployment process installs JEE application components in the JEE containers, as shown in Figure 4-5. An EJB container manages the execution of all enterprise beans for a single JEE application. Enterprise beans and their accompanying containers run on the JEE server. A web container manages the execution of all JSP and servlet components for a single JEE application. Web components and their accompanying containers run on the JEE server. An application client container manages the execution of all application client components for a single JEE application. Application clients and their accompanying containers run on the client machine. An applet container is the web browser and Java plug-in combination that runs on the client machine.

FIGURE 4-5

Application components and JEE containers



EXERCISE 4-1

JEE Architecture and the Nonfunctional Requirements of a System

Question Describe how JEE architecture affects the nonfunctional requirements of a system.

Answer The success of a software development project is dependent on many factors in addition to software functionality. It is important that you differentiate between functional and nonfunctional requirements. Nonfunctional requirements include the environment, platforms, tools, and networking components within which an application is implemented; they include quality-related issues such as scalability, speed of execution and response time, ease of installation, maintainability, and reliability. These nonfunctional requirements affect the capabilities of the functional requirements.

The JEE architect actively needs to account for all requirements, functional and nonfunctional, and needs to include all aspects of the project, including the packaging, installation, deployment, and maintenance of a software solution. Architects are often tasked with providing infrastructure design and layout for applications based on JEE technology. The JEE infrastructure provided by compliant and certified application servers typically offers techniques that meet nonfunctional requirements such as scalability, compatibility, and so on.

Nonfunctional requirements are specific. The ability to support a specified number of concurrent users, expected transaction throughput, maximum allowable response time, supported data growth rate, and acceptable end-to-end latency are important nonfunctional requirements that must be satisfied if the application is to be successful.

These JEE application server solutions have evolved to a point wherein nonfunctional requirements are addressed by built-in feature sets from these application infrastructure services. This allows application developers to focus their efforts on building functionality or business services. These applications use basic JEE services that are already built into servers sold by multiple vendors. For example, multithreading, concurrency handling, connection pooling, state/session synchronization, container-managed transactions, and persistence are feature sets of the application servers that address the nonfunctional requirements.

The nonfunctional requirements supported by JEE are divided into six categories:

- **Scalability** Concurrent connections, data growth rates, user-population growth rates, storage capacity, compute capacity, performance characteristics, and response-time requirements can be solved by connection pooling and application server clustering.
- **Security** Application-level security is handled by JEE via deployment descriptors, protection domains as well as network security, OS security, and database security.
- **Adaptability** Extensibility of the application; flexibility of the configuration; and the adaptive nature of the compute, storage, and network resources to changing demands from the application and application infrastructure are supported by JEE.
- **Compatibility** JEE provides multiplatform support (all UNIX, Win XP), cross-certification of application infrastructure solutions, multiple client devices, and back-end connectivity to legacy resources.
- **Manageability** Change management, problem management, asset management, and network/systems management.
- **Availability** Platform reliability, application infrastructure stability, and uptime requirements.

Development Methodology and Process

A JEE application is usually assembled from two different types of modules: enterprise beans and web components. Both of these modules are reusable; therefore, new applications can be built from preexisting enterprise beans and components. The modules are also portable, so the application that comprises them will be able to run on any JEE server conforming to the specifications. To build these modules, you will first need to consider designing the application using a modeling tool before using a development tool to implement code. The remainder of this section will take a look at each of these areas before moving into a discussion of what makes up a JEE application and the development phases of a JEE project.

Modeling Tools

Modeling is the visual process used for constructing and documenting the design and structure of an application. The model is an outline of the application, showing the interdependencies and relationships among the components and subsystems. Tools are available to facilitate this process, allowing you to show a high-level view of many objects. The Unified Modeling Language (UML) was created to unify the many proprietary and incompatible modeling languages that existed.

The use of modeling tools makes sense with the increasing complexity of Enterprise Java applications and components. However, learning to model comes from experience and from sharing knowledge about best practices and bad practices. Today, modeling involves the use and reuse of patterns. A *pattern* is commonly defined as a three-part rule that expresses a relationship between a certain context, a problem, and a solution. In other words, a pattern can represent a solution to a recurring problem or issue.

Development Tools

To be productive with technology such as JEE, analysts and programmers will inevitably need visual development tools for building JEE applications and components, e.g., Eclipse. When constructing a JEE application, a developer must not only create Java code but also build an archive file to house the classes and other supporting files, including XML deployment descriptors and reference resolutions. This archive must then be deployed to a server and tested. These sets of tasks will be repeated several times over before the application is finally ready to be deployed to a production environment. All of these tasks typically need to be coordinated among multiple developers. The tools available at this time are still maturing, and tool vendors frequently release newer versions of tools to ease the development process.

In addition to the tools themselves, application *frameworks* provide components and services based on the best patterns, practices, and standards available. The ideal framework would implement extendable design patterns on the presentation, business, and data/services layers. These implementations should work for any JEE-certified server.

The use of a framework or "best practice" may be helpful in preparing for and completing Part 2 of the SCEA exam. The following frameworks and guidelines are a non-exhaustive sample of what is available:

- **Spring framework** Supports all of the major JEE technologies, on all tiers (web, EJB, and data access). See *www.springframework.org* for more information.
- **Struts framework** An implementation of the Model View Controller (MVC) pattern. This classic framework can be used when developing web components consisting of JSPs and servlets. See *jakarta.apache.org* for more information.
- **Sun's JEE BluePrints (Pet Store)** A set of "best practice" guidelines for developing JEE applications. Along with the guidelines is a practical implementation of them, known as the Pet Store application. This application is the classic web shopping cart for buying a pet. This application is a good one to study and know before taking Part 2 of the exam, as it can provide some good working JEE code examples.

Contents of a JEE Application

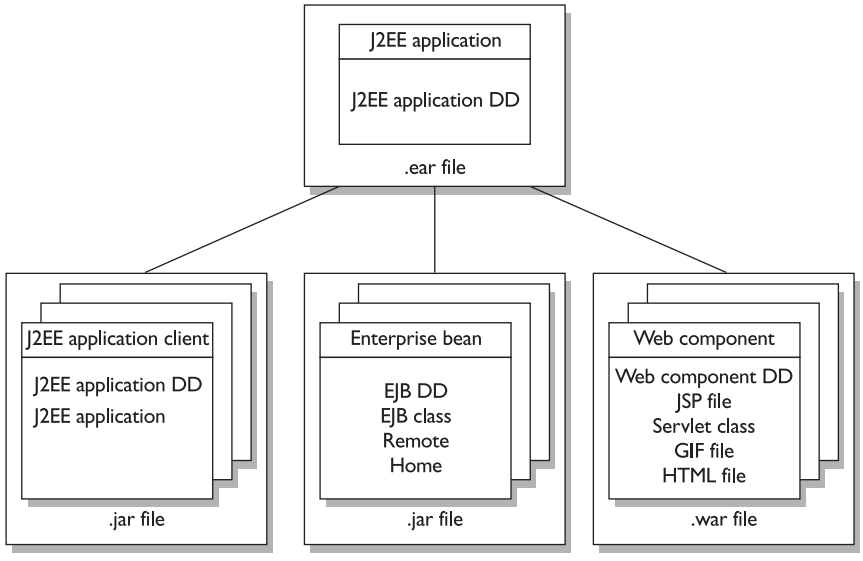
The hierarchy of a JEE application is shown in Figure 4-6. A JEE application may contain any number of enterprise beans, web components, and JEE application clients. The deployment descriptor in Figure 4-6 refers to a file that defines structural information such as class names, location, and other attributes to facilitate the deployment of the web or enterprise application.

Enterprise beans comprise three class files: the EJB class, the remote interface, and the home interface. Web components may contain files such as servlet classes, JSPs, HTML files, and GIFs. A JEE application client is a Java application, typically Java classes providing a user interface that runs in a container and is allowed access to JEE services.

Each JEE application, web component, and enterprise bean includes an XML file called a deployment descriptor (DD) that describes the component. An EJB DD has functions to declare transaction attributes and security authorizations for

FIGURE 4-6

Contents of a JEE application



an enterprise bean. This information is declarative; it can be changed without subsequent modification to the bean's source code. The JEE server reads this information at runtime, acting upon the bean accordingly. Each module is bundled into a file with a particular format, as seen in Table 4-1. While it won't affect the exam per se, it is important to note that the Java EE 5 platform introduces a simplified programming model and eliminates much of the boilerplate that earlier releases required. With Java EE 5 technology, DDs are optional. You can now enter the information as an annotation directly into a Java object. Annotations are a new feature, originally introduced in Java 2 Platform, Standard Edition (J2SE) 5.0. They are a form of metadata with a very simple syntax and recognizable. They begin with a leading at sign (@). Annotations are generally used to embed in a program data that would otherwise be furnished in a deployment descriptor. With annotations, you put the specification information right in your code next to the program element that it affects.

As the EISs repaired before the year 2000 refuse to go away, implementations that utilize user-developed or vendor-based APIs to access these systems with JEE are growing in popularity. To facilitate these APIs, a resource adapter is a JEE component that implements the JEE connector architecture (JCA) for a specific EIS. It is through the resource adapter that a JEE application communicates with an EIS. Stored in a Resource Adapter Archive (.rar) file, a resource adapter may be deployed on any JEE server, much like the .ear file of a JEE application.

TABLE 4-1

Files Used in JEE Applications

File Content	File Extension
JEE Enterprise Application	.ear
JEE application deployment descriptor	.xml
Enterprise JavaBeans	.jar
EJB deployment descriptor	.xml
EJB class	.class
Remote interface	.class
Home interface	.class
Web application component	.war
Web component deployment descriptor	.xml
JSP file	.jsp
Resource Adapter Archive file	.rar
Servlet class	.class
Image files	.gif and .jpg
HTML file	.html
JEE application client	.jar
JEE application client deployment descriptor	.xml
Java class	.class

Development Phases of JEE Applications

JEE applications may pass through the following developmental phases:

- Enterprise bean creation
- Web component creation
- Application assembly
- Application deployment

In larger organizations, separate individuals or teams may perform each of these phases. This division of labor is made more feasible by the creation of a portable file output by each phase. This file contains the input for the subsequent phase. In the optional enterprise bean creation phase, for example, a developer delivers EJB JAR files. In the web component creation phase, for example, a developer delivers web components in a WAR file. These phases are not sequential, but the assembly and

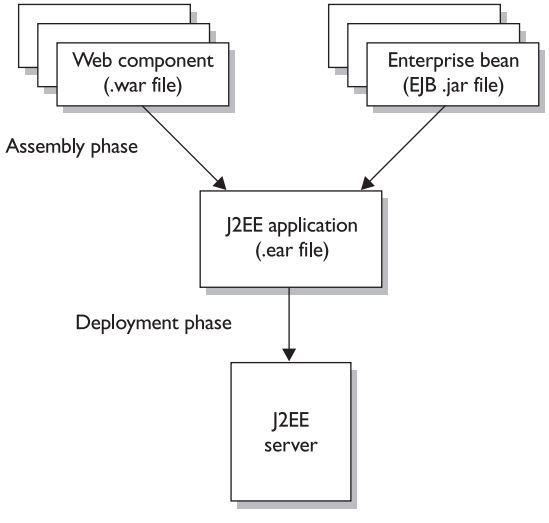
deployment phases are sequential and required. During the application assembly phase, another developer combines these files into a JEE application, saving it in an EAR file. Ultimately, a system administrator uses that EAR file to install the JEE application into an application server at deployment time. These final stages are illustrated in Figure 4-7.

Java and XML is a natural match for the creation of applications that exploit the web of information, where different classes of clients consume and generate information that is exchanged between servers that run on various system platforms. The portability and extensibility of both XML and Java make them the ideal choices for the flexibility and wide availability requirements of the web. The following sections summarize the development phases for JEE applications. Because a JEE application does not necessarily need both enterprise beans and web components, one of the first two phases is often skipped; all other phases are required.

Enterprise Bean Creation Enterprise bean creation is performed by software developers, who are responsible for coding and compiling the Java source code needed by the enterprise bean, specifying the deployment descriptor for the enterprise bean, and bundling the .class files and deployment descriptor into an EJB JAR file. That EJB JAR file is subsequently delivered to facilitate the next step. See Chapter 7 for more information.

FIGURE 4-7

Development phases of a JEE application



Web Component Creation Web component creation can be performed by web designers, who create the JSP components, along with software developers, who are responsible for the servlets. Java source code for the servlet is developed and compiled, JSP and HTML files are written, the deployment descriptor for the web component is specified, and the .class, JSP, HTML, and deployment descriptor files are bundled into the WAR file. That WAR file is delivered to facilitate the next step.

JEE Application Assembly The application assembler is the person who takes an EJB archive file (EJB JAR) and a web components archive file (WAR) and assembles them into a JEE Enterprise Archive, or EAR, file. The next step is to resolve any references, which include the following:

- Database connection pools
- Mail sessions
- URL connections
- JMS queues and topics
- EJB references

This process is handled by defining elements in one or more additional XML documents, also known as deployment descriptors or deployment plans. The assemblers or deployers can edit the deployment properties directly or use tools that add these XML tags. These additional files map internal references along with server-specific properties to JNDI or other names that exist in the destination JEE application server. The application assemblers perform the following tasks to deliver an EAR file containing the JEE application:

- Assemble EJB JAR and web components (WAR) files created in the previous phases into a JEE application (EAR) file.
- Specify the deployment descriptor for the JEE application.
- Verify that the contents of the EAR file are well formed and comply with the JEE specification.

The final deliverable for this stage is the completed EAR file containing the JEE enterprise application.

JEE Application Deployment The deployer is the person who configures and deploys the JEE application, administers the computing and networking

infrastructure where JEE applications run, and oversees the runtime environment. Duties include setting security attributes, setting transaction controls, and specifying database connection pools. During configuration, the deployer follows instructions supplied by the application component provider to resolve external dependencies, specify security settings, and assign transaction attributes. During installation, the deployer is responsible for moving the components to the server and generating the classes and interfaces specific to the destination container.

The deployer performs the following tasks to install and configure a JEE application:

- Stage the initial JEE application (EAR) file created in the preceding phase to the JEE server.
- Configure the JEE application for the operational environment by modifying the DD of the JEE application.
- Verify that the contents of the EAR file are well formed and comply with the JEE specification.
- Deploy (install) the JEE application EAR file into the JEE server.

CERTIFICATION OBJECTIVE 4.02

Explain the Use of Patterns in the JEE Framework

In the context of computer architecture, design patterns are proven solutions to recurring business problems. They consider the particular context of the problem and the consequences of the solution. A good designer will use a pattern because it is proven—that is, the designer has used it before successfully or has built and validated a proof of concept**.** Good architects use the experience, knowledge, and insights of developers who have used these patterns successfully in their own work. When a problem is common, a good designer doesn't have to devise a new solution; instead, he or she follows the pattern and adapts it to the current environment.

Use of Patterns in the JEE Framework

The JEE framework employs patterns to support these capabilities. These patterns will be covered in greater detail in Chapter 5, but to cover exam specifics, we will examine patterns that are prominent in the JEE framework. JEE uses the following core patterns

TABLE 4-2

Patterns Used with the JEE Framework

Pattern	Use	JEE Implementation
Proxy	Provides method calls to a principal object to occur indirectly through a proxy object that acts as an agent for the principal object, delegating method calls to that object	EJB remote interface
Decorator	Extends the functionality of a class such that it is transparent to its clients	*EJBObject*
Factory Method	Provides a reusable class independent of the classes it instantiates because it delegates the choice of which class to instantiate to another object; refers to the newly created object via a common interface	*EJBHome* interface
Abstract Factory	Provides a way to create instances of those abstract classes from a corresponding set of concrete subclasses	*EJBHome* interface

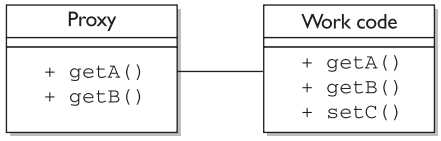
to enable flexible association of EJB classes with other components. The Proxy pattern provides a separate implementation of interface and working code for location transparency. The Decorator provides a similar contract for a class but with added functionality. The Factory Method provides ability to define a contract for creating an object but defers instantiation to subclasses. The Abstract Factory provides a contract for creating families of related or dependent objects without specifying concrete classes. Table 4-2 details the patterns used with the JEE framework.

The Proxy Pattern

The Proxy pattern (as shown in Figure 4-8) decouples interface and implementation by providing two objects. If the implementation class changes, the proxy remains the same. This is because the proxy interface includes only the method invocations and their signatures. So, it is lightweight in comparison to the implementation class (the working code). As architects, we typically examine what types of operations are expensive. For example, object creation and initialization is usually expensive. To improve application performance, it is a sound approach to defer object creation and object initialization to the time when you need the object. The Proxy pattern reduces the cost of accessing objects. It accomplishes this cost reduction because it

FIGURE 4-8

Proxy pattern



uses another object (the proxy) to act as a stand-in for the real object. The proxy creates the implementation object only if the user requests it. An example of the proxy pattern implemented in JEE is the EJB remote interface.

Decorator Pattern

The Decorator pattern provides the same contract for a class but with extended functionality. The pattern is used when functionality needs to be added to objects dynamically. The solution involves encapsulating the original object inside an abstract wrapper interface. Both the decorator objects and the base object inherit from this abstract interface. The interface is generic such that it allows a theoretically unlimited number of decorative layers to be added to each base object. Decorators would seem to be especially useful when you wish to add functionality when you do not have the actual code source of the class. If you know enough about the object—that is, the interface—when you want to decorate, you can provide a decoration for it.

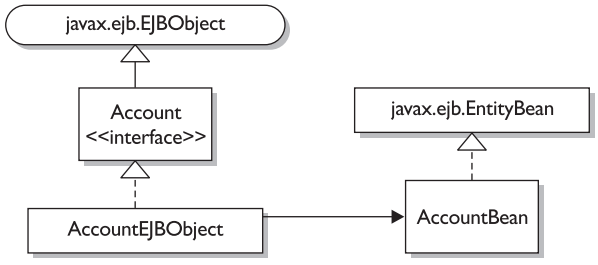
It is important to note that the decorator is a subclass of the component class that it decorates. This is counterintuitive because the instance is a parent of the component it decorates. Decorators share the "wrapper" approach. The difference, however, is intent. The decorator has value only if it changes the behavior of the "wrapee." The EJBObject is a decorator for the bean because the bean's functionality is expanded to include remote behavior.

Factory Method Pattern

The Factory Method pattern (as shown in Figure 4-9) provides the ability to define an interface for creating an object but defers instantiation to subclasses. JEE technology uses this pattern for the *EJBHome* interface, which creates new *EJBObjects*. (For more information, see Chapter 5 and perhaps consult the book *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides [Addison-Wesley, 1995].)

FIGURE 4-9

Factory Method



Abstract Factory Pattern

The Abstract Factory pattern provides an interface for creating families of related or dependent objects without specifying concrete classes. JEE technology uses this pattern for the *EJBHome* interface, which creates new *EJBObjects*.

EXERCISE 4-2

Patterns in the JEE Framework and Development

Question Describe the use of patterns in the JEE framework and development.

Answer A pattern is a recurring solution to a problem in a context. A *context* is the environment, surroundings, situation, or interrelated conditions within which something exists. A *problem* is an unsettled question, something that needs to be investigated and solved. A problem can be specified by a set of causes and effects. Typically, the problem is constrained by the context in which it occurs. Finally, the *solution* refers to the answer to the problem in a context that helps resolve the issues. In addition to the framework patterns used by JEE, patterns are used in development and are typically listed according to functionality. The presentation tier patterns contain the patterns related to servlets and JSP technology. The business tier patterns contain the patterns related to the enterprise beans technology. The integration tier patterns contain the patterns related to JMS and JDBC. Tables 4-3, 4-4, 4-5, and 4-6 contain partial lists of applicable patterns along with descriptions to provide a high-level overview of the patterns. The presentation tier patterns, business tier patterns, and integration tier patterns will all be discussed in detail in later chapters.

TABLE 4-3

Framework Patterns

Pattern Name	Description	JEE Implementation
Proxy	Provides method calls to a principal object to occur indirectly through a proxy object that acts as an agent for the principal object, delegating method calls to that object	EJB remote interface
Decorator	Extends the functionality of a class such that it's transparent to its clients	*EJBObject*
Factory Method	Provides a reusable class independent of the classes it instantiates, because it delegates the choice of which class to instantiate to another object, referring to the newly created object via common interface	*EJBHome* interface
Abstract Factory	Provides a way to create instances of those abstract classes from a corresponding set of concrete subclasses	*EJBHome* interface

TABLE 4-4

Presentation Tier Patterns

Pattern Name	Description
Decorating Filter	Facilitates pre- and post-processing of a request
Front Controller	Provides a centralized controller for managing the handling of a request
View Helper	Encapsulates logic that is not related to presentation formatting into Helper components
Composite View	Creates an aggregate View from atomic subcomponents
Service To Worker	Combines a Dispatcher component in coordination with the Front Controller and View Helper Patterns
Dispatcher View	Combines a Dispatcher component in coordination with the Front Controller and View Helper Patterns, deferring many activities to View processing

TABLE 4-5 Business Tier Patterns

Pattern Name	Description
Business Delegate	Decouples presentation and service tiers and provides a facade and proxy interface to the services
Value Object	Exchanges data between tiers
Session Facade	Hides business object complexity, and centralizes workflow handling
Aggregate Entity	Represents a best practice for designing coarse-grained entity beans
Value Object Assembler	Builds composite value object from multiple data sources
Value List Handler	Manages query execution, results caching, and result processing
Service Locator	Hides complexity of business service lookup and creation; locates business service factories

TABLE 4-6 Integration Tier Patterns

Pattern Name	Description
Data Access Object	Abstracts data sources; provides transparent access to data
Service Activator	Facilitates asynchronous processing for EJB components

CERTIFICATION OBJECTIVE 4.03

Describe the Concepts of "Best Practices" and "Guidelines"

Successful companies establish the use of refactoring, best practices, patterns, and tools; they spread the awareness of these among their JEE programmers and architects. Successful developers share their knowledge and pass on their proven techniques to others. The net result is productivity. The ultimate product is the implementation of solid applications.

The challenges we face in software development today as always are twofold:

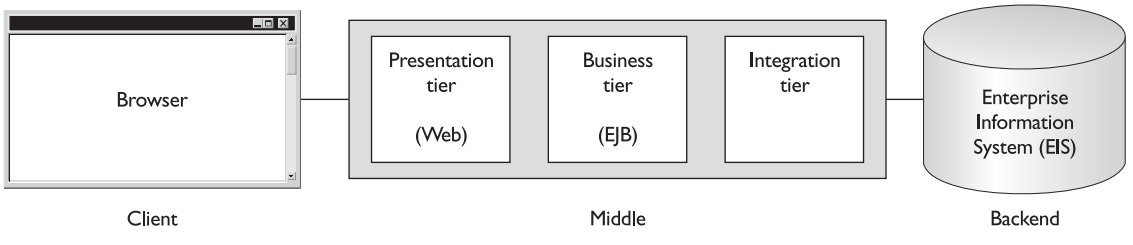
- Obtaining and maintaining the architectural and developmental skills to build effective enterprise systems
- Meeting market-driven time lines for developing new applications while maintaining quality in our implementations

Experienced architects often make trade-offs to meet deadlines. These trade-offs include the use of refactoring and other techniques to optimize development time and address the inherent performance considerations. We tend to hold onto our software designs for too long. Even after they become unwieldy, we continue to use and reuse code that is no longer maintainable—because it works, we are afraid to modify it. But this is typically not cost effective. When we refactor, we remove redundancy, eliminate unused functionality, and reengineer designs. Refactoring throughout the entire project life cycle saves time and increases quality. Refactoring keeps the design simple, and avoids needless complexity, keeps code clean and concise, and makes it easier to understand, modify, and extend the code. Moreover, application behavior is known to change throughout the application life cycle—that is, from development to production. As performance and stability issues are discovered, the application should be amended and improved.

Experience yields BluePrints for solving recurring problems: therefore, the term *best practice*. A best practice is a technique used to drive design at the component level. For example, a best practice might be to use session beans as facades to entity beans. On the other hand, a *guideline* is a rule applied horizontally to the design. For example, to minimize network traffic, the architect attempts to maximize the content of data requests—that is, we try to get as much as we can out of each request.

As mentioned, the JEE architecture typically consists of three basic tiers in the platform, as shown in Figure 4-10. With respect to each tier, we will introduce and review some of the best practices.

FIGURE 4-10 JEE tiers



Identifying the Need for Best Practices

In development, the question should not be "Is there a best practice?" Instead, it should be "Will this best practice improve our application?" For example, caching of frequently used data and references will benefit any JEE application; however, determining what is *frequently used* can be difficult in development. By testing early releases of an application, developers can locate and correct inappropriate design decisions.

In the quality assurance (QA) stage of an application life cycle, it is often assumed that best practices and appropriate patterns have already been applied. The typical QA team is unlikely to have the expertise needed to identify the need for best practices. If the application is deemed to perform poorly or fail, it will have to be amended by the development team at a great expense. It is therefore extremely important to determine the application hot spots and suggest the primary candidates for reworking.

In the enterprise production environment, applying new design patterns and best practices to a production system is more than difficult—both technically and politically. It can, however, be essential to creating a mature and ultimately successful application. Interrupting service to clients to redeploy a production application will commonly happen if the application fails or is unusable. What the good architect has is the uncommon ability to identify when performance is under par and justify expending additional resources to refactor by specifying the nature of the problem.

You can apply best practices and guidelines in each tier, including the client tier, web tier (presentation), EJB tier (business logic), and database or EIS integration tier (integration). You can also apply refactoring and guidelines to orthogonal services that span tiers including security and transaction processing.

Best Practice: Client Tier

The client tier serves as an interface with any number of other systems. A transaction enters the workflow, for example, as an HTML request from a standard web browser or as an XML-formatted electronic message from another system. You should decouple the client type from the enterprise application by using an HTML browser, an applet, a Java application, and, last and but certainly not least, a non-Java application.

CERTIFICATION OBJECTIVE 4.04

Illustrate the Use of JEE for Workflow

A common method for designing applications is to organize them around an event-driven user interface. In this design pattern, the developer creates the interface and subsequently writes code that will execute the desired application actions in response to user gestures. This structure can be successful for small, single-user systems that will require little alteration to the functionality over time. However, it is not suitable for larger, distributed projects for the following reasons:

- More sophisticated applications may require data to be viewed and manipulated in several different ways. When business logic is implemented in the display code, display inconsistencies can result because the logic can be copied and modified in one object and not another. In addition, any change to the data display requires updates in several places.
- When data manipulation logic, format and display code, and user event handling are entangled, application maintenance can be very difficult, especially over a long span of time.
- User interfaces cannot be reused if the application logic has been combined with the code for an existing interface.
- Added functionality may require several changes to existent code, which may be difficult to locate.
- Business logic code may access a vendor-specific product (a database, for example), thus making the application much less portable.
- Changes to a single piece of code may have far-reaching side effects.
- Development cannot occur on a modular basis, as everything is dependent on everything else. This problem is amplified on large-scale development projects because it is difficult for a large team of developers to split tasks.
- Code is less reusable, because components are dependent on one another; therefore, they are less usable in other contexts.

To overcome these shortcomings, utilizing the MVC design pattern best practice results in a separation of the application data from the ways that the data can be accessed or viewed as well as from the mapping between system events (such as user interface events) and application behaviors.

Best Practice: MVC Pattern

The MVC pattern consists of three component types:

- **Model** Represents the application data along with methods that operate on that data.
- **View** Displays that data to the user.
- **Controller** Translates user actions such as mouse movement and keyboard input and dispatches operations on the Model.

As a result, the Model will update the View to reflect changes to the data. Figure 4-11 illustrates the functions of each of these component types as well as the relationships among them.

When this best practice is used properly, the Model should have no involvement in translating the format of input data. This translation should be performed by the Controller. In addition, the Model should not carry any responsibility for determining how the results should be displayed.

Table 4-7 displays the participants and collaborations involved with the three components.

The MVC model should be used as follows:

- For distributed applications
- For larger applications
- For applications with a long lifetime
- Where interface and back-end portability are important

FIGURE 4-11

Relationships among components

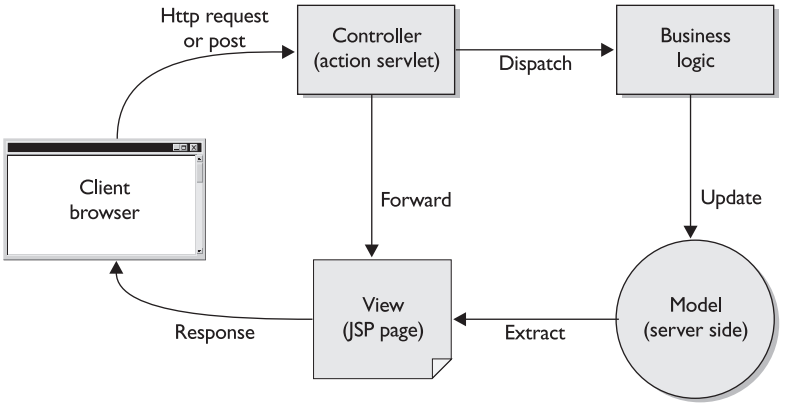


TABLE 4-7 MVC Participants and Collaborations

Component	Participants	Collaborations
Model	Extrapolates the business logic of the application Maintains the application state Provides access to functions of the application Manages persistence Extracts data for the View component Informs interested parties of changes to data	Informs the View when it makes changes to the application data Can be queried by the View Gives the Controller access to application functionality
View	Extrapolates data presentation Responds to users with data Maintains consistency with Model data	Provides Model data for the user Refreshes display when informed of data changes by the Model Transfers user input to the Controller
Controller	Extrapolates user interaction/application semantic map Transforms user actions into application actions Determines the appropriate data display based on user input and context	Transforms user inputs and dispatches class logic or application actions on the Model Selects the View to present based on user input and Model action outcomes

- Where data must be viewed and manipulated in multiple ways
- To facilitate maintenance
- To support simultaneous, modular development by multiple developers
- To allow division of labor by skill set
- To facilitate unit testing
- When employing enterprise beans that are reusable across applications

MVC offers the following benefits:

- Clarifies application design through separation of data modeling issues from data display and user interaction
- Allows the same data to be viewed in many ways and by many users
- Simplifies impact analysis, thereby improving extensibility
- Facilitates maintenance by encapsulating application functions behind trusted APIs

- Enhances reusability by separating application functionality from presentation
- Facilitates distribution of the application, as MVC boundaries are natural distribution interface points
- Can be used to divide deployment as well as make incremental updates possible
- Forces clear designation of responsibilities and functional consistency, thereby facilitating testability
- Increases flexibility, because data model, user interaction, and data display can be made "pluggable"

MVC designs may encounter the following problems:

- *Components aren't able to take advantage of knowledge of other components' implementation details*. This may have a negative effect on application performance. Skillful API design that optimizes the length of the code path (number of machine cycles) for each API function can assist in avoiding this problem to some extent.
- *Communication volume and other latency issues must be carefully addressed; otherwise, MVC may not scale well in distributed systems*. Latency comes from several sources. Web application servers may take some time to process a request, especially if they are overloaded and model components are not local. Web clients can add delay if they do not efficiently handle the retrieved data and display it for the user. Latency caused by client or sluggish servers, however, can in principle be solved simply by providing a faster server or clustering.
- *Maintenance of an MVC application may be difficult if the Model API is unstable, because the Controller is written in terms of the Model API*. There should be a decoupling between the sender and the receiver. A sender is an object that invokes an operation, and a receiver is an object that receives the request to execute a certain operation. The term *request* here refers to the command that is to be executed. This also allows us to vary when and how a request is fulfilled. This decoupling provides us with flexibility as well as extensibility. The command pattern turns the request into an object that can be stored and passed around in the same way as other objects. This provides a hook for Controller extensions to handle new Model functions. In addition, an adapter can often provide backward API compatibility.

MVC and the Struts Framework

The Struts framework has been developed by the Jakarta Project, which is sponsored by the Apache Software Foundation, to provide an open-source framework for building web applications with Java Servlet and JavaServer Pages (JSP) technology. The Struts framework is not the most popular framework for new development but it is probably the most prevalent. Struts supports application architectures based on the MVC design paradigm. The official Struts home page can be found at *http://jakarta.apache.org/struts*.

The primary areas of functionality included in Struts are:

- **A controller servlet** Dispatches requests to appropriate Action classes provided by the application developer
- **JSP custom tag libraries** Facilitate creation of interactive, form-based applications
- **Utility classes** Provide support for XML parsing, automatic population of JavaBeans properties, and internationalizing prompts and messages

Struts applications adhere to the MVC design pattern. The three major components are the servlet Controller, JavaServer Pages (the View), and the application's business logic (the Model), as shown in Figure 4-12.

The following text describes the process illustrated in Figure 4-12.

First, the user request goes to a Controller that initializes the context/session for the overall transaction and alerts Page A's Model. The Controller then forwards execution to Page A's View (JSP).

Page A posts back to the Controller, which calls Model A to validate the posted form data. If the input is invalid, Model A returns a result that the Controller uses to forward/redisplay Page A. The entire *HttpServletRequest* might be made available to the Model A bean, or the Controller might be clever enough to call setters for each posted form field via introspection.

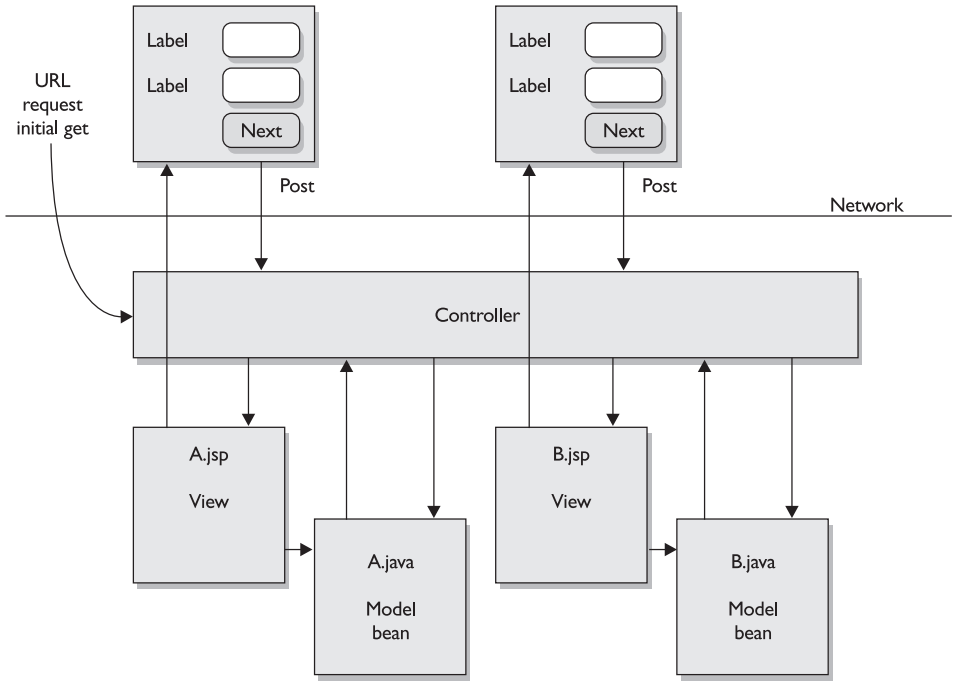
The Controller determines who is next in the chain through use of a multitude of options: straight if-else (or switch-case) code, an XML document, database records, or a rules engine.

The Controller is the centralized "traffic cop" that knows when all required steps in a transaction are complete. The Model preserves the "state" (posted form fields) and holds validation logic. If the user action is invalid, the Controller is alerted to redisplay the same View/JSP.

The Controller bundles and directs HTTP requests to other objects in the framework, including JSP. After it has been initialized, the Controller parses

FIGURE 4-12

Simple MVC/ Struts example flow



a configuration resource file. This resource file defines the action mappings for the application. These mappings are used by the Controller, turning HTTP requests into application actions.

At the very least, a mapping must specify a request path as well as what object type is to act upon the request. The action object either handles the request and responds to the client (usually a web browser) or indicates that control should be forwarded to another action.

Because action objects are linked to the application's Controller, they have access to that servlet's methods. When an object is forwarding control, it can indirectly forward one or more shared objects, such as JavaBeans, by putting them in a standard collection shared by Java Servlets.

Most of the business logic in a Struts application can be represented using JavaBeans. In addition, JavaBeans can be used to manage input forms, eliminating the common problem with retaining and validating user input between requests. Using Struts, data can be stored in a form bean. The bean can then be saved in a shared context collection, making it accessible to other objects, especially action objects.

It could be used by a JSP to collect data, by an action object to validate user input, and then by the JSP again to repopulate the form fields.

In the case of validation errors, Struts has a shared mechanism for raising and displaying error messages. Struts form beans are defined in the configuration resource and then linked to an action mapping via a common property name. When a request calls for an action that utilizes a form bean, the controller servlet will either retrieve the form bean or create one. That form bean is passed to the action object. The action object checks the contents of the form bean before its input form is displayed, queuing messages that are to be handled by the form. When the action object is ready, it can return control by forwarding to its input form, which is usually a JSP. The controller is then able to respond to the HTTP request, directing the client to the JSP.

Custom tags are included within the Struts framework. They have the ability to automatically populate fields from a form bean. Most JSPs must know only the proper field names and where to submit the form. Components such as messages set by the action object can be output using a single custom tag. Application-specific tags can be defined to hide implementation details from the JSP pages.

These custom tags are designed to use the Java platform internationalization features. All field labels and messages can be retrieved from a message resource, with Java automatically providing correct resources for the client's country and language. Providing messages for another language requires only the addition of another resource file.

Other benefits of custom tags are consistent labeling between forms and the ability to review all labels and messages from a central location.

In most applications, action objects should pass requests to other objects, primarily JavaBeans. To enable reuse on other platforms, business logic JavaBeans should not refer to any web application objects. Action objects should translate required details from the HTTP request, passing those along to business-logic beans as regular Java variables.

In database applications, business-logic beans may connect to and query the database, returning the result set to the action's servlet, where it is stored in a form bean and then displayed by the JSP.

The Model

The Model in an MVC-based application can be divided into two parts: the internal state of the system and the actions that can be taken to alter that state.

A web application using the Struts framework typically represents the internal state of the system as a set of one or more JavaBeans with properties that represent the details of that state. These beans may be self-contained and able to save their

state information persistently. Additionally, they may be facades that know how to retrieve information from external sources such as databases when that information is requested. Entity EJBs (Enterprise JavaBeans) can also be used to represent internal state.

Larger applications often represent possible business logic actions for a system as methods. Other systems may represent possible actions separately, often as Session EJBs. Smaller-scale applications may embed the available actions within the Action classes that are part of the Controller role. This is effective only when the logic is simple and reuse is not an issue. It is a good idea always to separate the business logic from the roles of the Action classes.

The View

Struts-based applications generally utilize JSPs to construct the View component. The JSP environment includes a set of standard action tags such as `<jsp:useBean>` as well as the ability to define custom tags and organized custom tag libraries. In addition, it is sometimes necessary for business objects to render themselves in HTML or XML, depending on their state at request time. It is easy to include the output from these objects in a resulting JSP page by using the `<jsp:include>` standard action tag.

The Controller

The Controller portion of the application focuses on receiving requests from the client (most often a user running a web browser), deciding what business logic function is to be performed, and delegating responsibility for producing the next phase of the user interface to an appropriate View component. Struts utilizes a servlet of class *ActionServlet* as the main component of the Controller. This servlet is configured through definition of a set of mappings that are described by a Java class *ActionMapping*. Each mapping defines a path, matched against the request URI of the incoming request as well as the fully qualified class name of an Action class. The Action class is responsible for performing the desired business logic and subsequently dispatching control to the appropriate View component to create the response.

In addition, Struts supports the ability to use *ActionMapping* classes with additional properties beyond the standard ones needed to operate the framework. This enables storage of additional application-specific information while utilizing the remaining features of the framework. Furthermore, Struts allows definition of logical names for the forwarding of the control. The method can ask for the logical name of the page without knowing the actual name of the corresponding JSP page.

MVC and other Frameworks

As Struts popularity has peaked, a number of new MVC Frameworks have emerged. For example:

- **JavaServer Faces (JSF)** One of the most popular MVC Frameworks. It uses a component-based architecture making it the best supported framework for pluggable third-party visual controls. See *http://java.sun.com/javaee/javaserverfaces* for more information.
- **WebWork** An open-source MVC Framework that it is being integrated into Struts as Struts II. See *www.opensymphony.com/webwork/* for more information.
- **Tapestry** An open-source MVC Framework that emphasizes simplicity of development and scalability of applications. See *http://tapestry.apache.org* for more information.

EXERCISE 4-3

Structuring Development

Question When building a JEE application within an enterprise where valuable legacy systems exist and Java with JEE is growing in use but not the strong suite in the skill set of the current majority of the enterprise developers, how can you structure the development to maximize productivity?

Answer Productivity with a new technology can generally be achieved by striving for the following goals:

- **Code and design reuse** Maximize code reuse so as to decrease the cost for development, provide incremental quality improvements, and establish design best practices that everyone in the organization understands.
- **Rational functional decomposition** Every class in the design should play a clearly defined role in the application. The resulting design clarity will facilitate maintenance, impact analysis and system extension, and flatten the learning curve for new developers.
- **Development tasks isolated by skill sets** The design should partition the application into chunks that reflect the skill sets of subteams in the

development group. For example, the design specifies using JSP tag libraries, instead of JSP pages with embedded code, and then web page designers with no knowledge of programming can operate in parallel with programmers, and programmers can focus on solving coding problems.

- **Decouple classes with differing rates of change** Design such that parts of the application that change quickly will require both ease of change and looser coupling to the rest of the system. Subsystems that change more slowly can be more tightly coupled, providing efficiency opportunities.
- **Extensibility** The application functionality must be able to keep up with organizational growth and technological change.
- **Modularity** We should break the design into modules that interact through well-defined interfaces, thereby allowing the developer to work independently, enhance maintainability and testability, and provide opportunities for using purchased components and outsourcing some development.
- **Security** Data security enforcement is crucial, especially if the application is performing financial transactions, for the privacy and security of customers.
- **Common look-and-feel** The application GUI should be designed such that the user can always intuitively know where to look for desired information.
- **Minimize network traffic** The application should avoid transmitting data needlessly or redundantly.
- **Allow for multiple user interfaces** The data should be represented in a way most appropriate for the task at hand. New types of user interfaces should be easy to add.
- **Persistent data must always be consistent** The design should fulfill the goal by using the MVC design pattern to separate form, function, and data; by dividing the application into functional modules and multiple tiers; and by applying several design patterns, which are common problem solutions that have been found to work well in the past. The existing legacy systems can be accessed via the JEE components by using the Proxy, Adapter, or Facade patterns to "wrap" legacy systems with Java-based APIs such as JNI (Java Native Interface) so that legacy developers can continue their work and JEE developers can interface with the technology, obviating the need to rebuild the legacy functionality in JEE. The legacy systems are given a new life, as they are now JEE-enabled.

EXERCISE 4-4

Defining Best Practice and Guideline

Question Define the concepts of *best practice* and *guideline*.

Answer A best practice is a technique used to drive design at the component level. A best practice is an optimal process that is recognized by peers in similar situations. It is applicable to a cross-section of scenarios with varying resources and sizes. It takes design requirements into consideration. For example, a best practice might be to use session beans as facades to entity beans.

On the other hand, a guideline is a rule applied horizontally to the design. Guidelines reflect agreements on practices or operations by recognized professional associations. This includes formal, approved standards, as contrasted to de facto standards and proprietary standards that are exceptions to this concept. For example, to minimize network traffic, the architect attempts to maximize the content of a data request—that is, we try to get as much as we can out of each request.

CERTIFICATION OBJECTIVE 4.05

Review Best Practices Applicable for All Tiers

The MVC best practice is a design pattern that can be applied across all tiers. The MVC architecture is a way to divide functionality among objects involved in maintaining and presenting data so as to minimize the degree of coupling between the objects. The MVC architecture was originally developed to map the traditional input, processing, and output tasks to the graphical user interaction model. However, it is straightforward to map these concepts into the domain of multi-tier web-based enterprise applications.

In the MVC architecture, the Model represents application data and the business rules that govern access and modification of this data. The Model can be represented in many ways, including but not limited to EJBs and JavaBeans. The Model notifies views when it changes and provides the ability for the View to query the Model

about its state. It also provides the ability for the Controller to access application functionality encapsulated by the Model. The View renders the contents of a Model. It is usually implemented as a JSP. It accesses data from the Model and specifies how that data should be presented. When the Model changes, it is the responsibility of the View component to maintain consistency in its presentation. The View forwards user gestures to the controller.

A Controller defines application behavior; it interprets user gestures and maps them into actions to be performed by the model. The servlet best fits this task. In a stand-alone GUI client, these user gestures could be button clicks or menu selections. In a web application, they appear as `GET` and `POST` HTTP requests to the web tier. The actions performed by the Model include activating business processes or changing the state of the Model. Based on the user gesture and the outcome of the Model commands, the Controller selects a View to be rendered as part of the response to this user request. Usually, one Controller exists for each set of related functionality.

The implementation of MVC pattern offers the following benefits:

- Clarifies application design through separation of data modeling (Model) issues from data display (View) and user interaction (Controller)
- Facilitates distribution of the application, as MVC boundaries are natural distribution interface points

CERTIFICATION OBJECTIVE 4.06

Review Best Practices for the Client Tier

Thin-client solutions (HTML on a browser) are important to Internet-based applications. The browser acts as your client for rendering the presentation as encoded in HTML.

In addition to content that can be rendered with static HTML, the following items can be used to create web content: JSPs, servlets, applets, and JavaScript can be used to enhance the browser interface.

CERTIFICATION OBJECTIVE 4.07

Enumerate the Components and Categories of the Web Tier

The web tier produces responses that can be handled by the use of *web component*, a software entity that provides a response to a request. A web component typically generates the user interface for a web-based application. The JEE platform specifies two types of web components: servlets and JSP pages.

Web components are hosted by servlet containers, JSP containers, and web containers. In addition to standard container services, a *servlet container* provides network services by which requests and responses are sent, decodes requests, and formats responses. All servlet containers must support HTTP as a protocol for requests and responses, but they may also support additional request-response protocols such as HTTPS. A *JSP container* provides the same services as a servlet container and an engine that interprets and processes a JSP page into a servlet. A *web container* provides the same services as a JSP container and provides access to the JEE service and communication APIs.

CERTIFICATION OBJECTIVE 4.08

Explain How to Apply MVC to the Web Tier

The MVC design pattern can be applied to the web tier because it results in a separation of the application data from the ways that the data can be accessed or viewed as well as from the mapping between system events (such as user-interface events) and application behaviors.

As mentioned, the MVC pattern consists of three component types. The Model represents the application data along with methods that operate on that data. The View component displays that data to the user. The Controller translates user actions such as mouse movement and keyboard input and dispatches operations on the Model. As a result, the Model will update the View to reflect changes to the data. The View consists of JSP amended with JavaScript and embedded tags that can provide the full function user interface. The Controller is usually a servlet, and the Model can be a JavaBean or an EJB.

CERTIFICATION OBJECTIVE 4.09

Review the Best Practices for the Presentation Layer

To avoid needlessly complex presentation components in the web tier, follow these practices:

- Separate HTML from Java.
- Try to place business logic in JavaBeans.
- Factor general behavior out of custom tag handler classes.
- Favor HTML in Java handler classes over Java in JSPs.
- Use an appropriate inclusion mechanism.
- Use a JSP template mechanism.
- Use style sheets.
- Use the MVC pattern.
- Use available custom tag libraries.
- Determine the appropriate level of XML compliance.
- Use JSP comments in most cases.
- Follow HTML best practices.
- Utilize the JSP exception mechanism.

EXERCISE 4-5

Illustrate the Use of JEE for Workflow

Question In a web application, what type of component is usually used for the View and Controller elements of the MVC pattern? What type of component is usually used for the Model element?

Answer Most applications implementing the MVC generally utilize JSPs to construct the View component. Most applications generally utilize a servlet as the main component of the Controller. An application typically represents the Model, which is the internal state of the system, as a set of one or more JavaBeans with properties that represent the details of that state. These beans may be self-contained,

and they are able to save their state information persistently. Additionally, they may be facades that know how to retrieve information from external sources such as databases when that information is requested. Entity EJBs can also be used to represent internal state.

CERTIFICATION OBJECTIVE 4.10

Review the Internationalization and Localization

To operate in a global economy, JEE information systems must address a number of additional requirements, including the following:

- **Language requirements** Users of a globally available application may speak different languages. The relationship between geographic region and language spoken is not simple. Representation of such quantities as numbers, dates, times, and currency vary by region.
- **Legal differences** Countries vary in customs law and information privacy requirements. Some governments place limitations on ideas, images, or speech.
- **Financial considerations** Currencies are not necessarily freely convertible. Forms of payment may differ; for example, not all customers can be assumed to have a credit card or purchase order number. Governments have different requirements for customs restrictions, tariffs, and taxes.

Internationalization terminology is commonly used inconsistently, even within the internationalization field. This section presents definitions of common internationalization terms as they are used in the rest of the chapter.

Internationalization, Localization, and Locale

The set of location-specific elements represented in an application is called a locale. To be effective, applications should customize data presentation to each user's locale. Internationalization is the process of separating locale dependencies from an application's source code. Interestingly, internationalization is also known as "I18n" because the first character is I, and between the first and last character there are 18 characters with a last character of *n*. Examples of locale dependencies include messages and user interface labels, character sets, encoding, and currency

and time formats. Localization (also called "L10n") is the process of adapting an internationalized application to a specific locale. An application must first be internationalized before it can be localized. Internationalization and localization make a JEE application available to a global audience.

An internationalized JEE application does not assume the locale. If requests from clients arrive with an associated locale, then the response should be tailored for the locale. Internationalizing an existing application requires refactoring. Internationalization is fundamentally an architectural issue. Internationalization is facilitated if it is integrated into the application design. A project's design phase should identify and separate locale dependencies if the application might ever need to support multiple locales.

EXERCISE 4-6

Localization and Internalization

Question Describe the use of *localization* and *internalization*.

Answer Internationalization, also known as *I18n*, is the process of separating locale dependencies from an application's source code. Examples of locale dependencies include messages and user interface labels, character sets, encoding, and currency and time formats. Localization, also called *L10n*, is the process of adapting an internationalized application to a specific locale. An application must first be internationalized before it can be localized. Internationalization and localization make a JEE application available to a global audience.

The EJB Tier

The EJB tier hosts the application-specific business objects and the system-level services (such as transaction management, concurrency control, and security). The EJB tier is a critical link between the web tier and the EIS integration tier. It typically hosts the entity beans and session beans, data access objects and value objects, and perhaps master-detail modeling using enterprise beans.

JEE Best Practices: Data Access Objects

Unfortunately, most systems in use today rely on specific features of the enterprise's standard system resources, such as a vendor DBMS; they merge business logic and data access mechanisms. The result is lack of portability. As these standard resources become obsolete, the application systems tied to a resource become a real chore to upgrade. A good architect wants to avoid tying an application's business logic components to a resource, so the architect upgrades to a system with the least amount of resistance.

The data-access object (DAO) pattern separates the interface to a system resource from the code used to access that resource by encapsulating the access to the data. Each enterprise bean that accesses a back-end resource in the application may have an associated DAO class, which defines an abstract API of operations on the resource. This allows a clean separation of bean and database access code. This also ensures easier migration to and from bean to container-managed persistence for entity beans and allows for cross-database and cross-schema capability. This abstract API makes no reference to how the resource is implemented. The DAO simply has to know how to operate from the persistent store given some identity information such as a filename. For example, an enterprise bean uses data it obtains from the DAO, not directly from the database. In this way, the enterprise bean defers its persistence mechanism to the DAO, allowing it to concentrate entirely on implementing business methods.

JEE Best Practices: Value Objects

Some enterprise information objects have values that are used together. For example, in a shopping cart application, the fields of the Address object are always used together. Using a complete remote interface for such entity beans is overkill and results in unacceptably high server communication.

The data for an Address object can be retrieved once, sent to the client from the server in serialized form, and instantiated on the client. From then on, the local copy of the Address information can serve as a proxy for the Address property of the remote Order object. Subsequent accesses to the Address object's state are local, require no server communication, and use fewer resources. If the Address object is updated and sent to the server to update the server-side object, the entire object is sent. Furthermore, local accesses obviously have lower latency than accesses deferred through a remote interface.

Such a client-side object is called a *Value Object,* because it represents a composite value from the server, not a reference to an object on the server. Value Objects tend to be more-or-less ad hoc groupings of data values to support a use case (or group of use cases).

Use Value Objects for business objects that represent structure with accessor get and set behavior only.

You should use a Value Object when the business entity being modeled has

- Only methods that get values from the object's internal state (that is, immutable state)
- A life cycle that is completely controlled by another object
- A relatively small size

Whenever you update a bean by passing it a Value Object, the code should inspect all attributes of the Value Object and update the corresponding model bean attributes with their values. It should also check whether the version number of the Value Object is different from the model bean's version number. If this is the case, it should throw an exception, which indicates that the bean has been updated by another client.

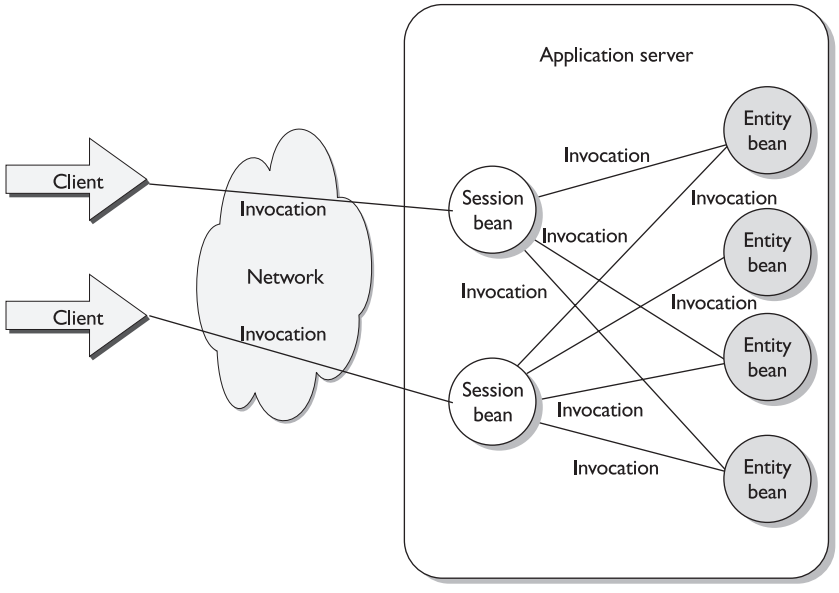
JEE Best Practices: Session Bean Facade

The session bean facade (SBF), shown in Figure 4-13, provides a simple, single point of entry to shared entity beans. It shields the client from complex entity bean relationships. The most obvious rationale for using session beans to abstract entity beans is that the approach also abstracts the structure of your data stores. The presumption is that you do not want to expose the inner workings of your application's data store (such as the database tables and columns), or even the specifics of how that data is stored. In other words, letting users (potential hackers) know your database schema is a not a good idea. Problems can arise when you allow direct access to the entity bean layer.

The methods in entity beans typically map directly to underlying fields in the data schema. This will become more important as service-based computing increases. Instead of providing complete applications, the JEE specification (or Web Services: UDDI, SOAP) indicates that organizations are focusing more on components than on complete applications. Interchanging data components from enterprise A's application with presentation components from enterprise B's application is

FIGURE 4-13

Session bean facade



becoming the standard. As a result, it is unsafe to assume that only your enterprise will be accessing your business layer and EJBs. For these reasons, a sound design of the business layer can save trouble when beans you worked on must be accessible by a new business partner.

By interjecting a layer of indirection in the form of a session bean, these problems were easily solved (as shown in Figure 4-13). The session beans become responsible for determining user permissions, providing greater flexibility and reuse. Session beans perform collections of calls to the entity beans on behalf of the remote clients, reducing network traffic. The transactional scope can be applied to methods that call groupings of entity beans, thus reducing the transactional overhead. Client-tier code requirements are simplified as more business logic is executed on the server.

JEE Best Practices: Master Detail

In a master-detail relationship, one object serves as a parent node to another. A master-detail relationship is a one-to-many type relationship. For example, if you receive an order and a set of items placed for each order, a master-detail relationship

is created by having the order number as a common field between the two. An application can use this master-detail relationship to enable users to navigate through the order data and see the detailed item data for orders placed.

When modeling a master-detail relationship as enterprise beans, the guidelines for using entity or session beans still hold. The choice is not affected by the master-detail relationship. However, the relationship is relevant when designing the behavior of the master. For example, suppose the master object should be modeled as a session bean and the details object should be an entity bean. In analyzing various possible combinations of session beans, entity beans, or value objects to represent master and detail objects, these questions are relevant only when the details are entity beans. For this case, two scenarios are possible:

- If the client modifies the detail entity object, the master object needs to expose the underlying entity object to the clients.
- If the client does not modify the detail entity object, the master object can have the necessary business logic to know which detail bean to access to construct the logical master-detail object. The client should not be exposed to the logic associated with accessing and aggregating the entity beans representing the details.

EXERCISE 4-7

Data Access Objects

Question Define data access objects and describe their purpose.

Answer The DAO pattern separates the interface to a system resource from the code used to access that resource. The DAO class defines an abstract API of operations on the resource. The DAO knows how to operate from a persistent store, based on some identity information such as a filename. The enterprise bean defers its persistence mechanism to the DAO, allowing the EJB to concentrate entirely on implementing business methods. Use the DAO to encapsulate access to data, maintain clean separation of bean and database access code, ensure easier migration to container-managed persistence for entity beans, and allow for cross-database and cross-schema capability.

EXERCISE 4-8

Value Objects

Question Define Value Objects and describe their purpose.

Answer A Value Object represents a composite value from the server, not a reference to an object on the server. Value Objects are ad hoc groupings of data values to support a use case (or group of use cases). Value Objects can be used for fine-grained business objects that represent structure with get/set behavior only. Use a Value Object when the business entity being modeled has

- Only methods that get values from the object's internal state (that is, immutable state)
- A life cycle that is completely controlled by another object
- A relatively small size

EXERCISE 4-9

Facades

Question Describe the use of session bean facades and their purpose.

Answer The SBF provides a simple, single point of entry to shared entity beans. It shields the client from complex entity bean relationships. SBF manages workflow on the client's behalf, and it reduces remote calls to the server. Architects using EJB technologies discovered almost immediately that providing access to entity beans from the client layer presents multiple problems, such as an overabundance of network traffic and latency, awkward security management, inefficient transactional behavior, and limits in reusability.

JEE Best Practices: EIS Integration Tier

The EIS Integration tier provides the information infrastructure for an enterprise. Accessing EIS can be complex, requiring vendor-specific knowledge of the following:

- Application programming model
- Transactions
- Security

JEE reduces the complexity of accessing an enterprise information system by relying on the web and EJB containers to handle transactions, security, and scalability. JDBC accesses relational data. JNDI accesses enterprise name and directory services. JMS sends and receives messages using enterprise messaging systems. JavaMail sends and receives mail. JavaIDL calls CORBA services. JNI calls services written in other languages—JNI can interact with native languages.

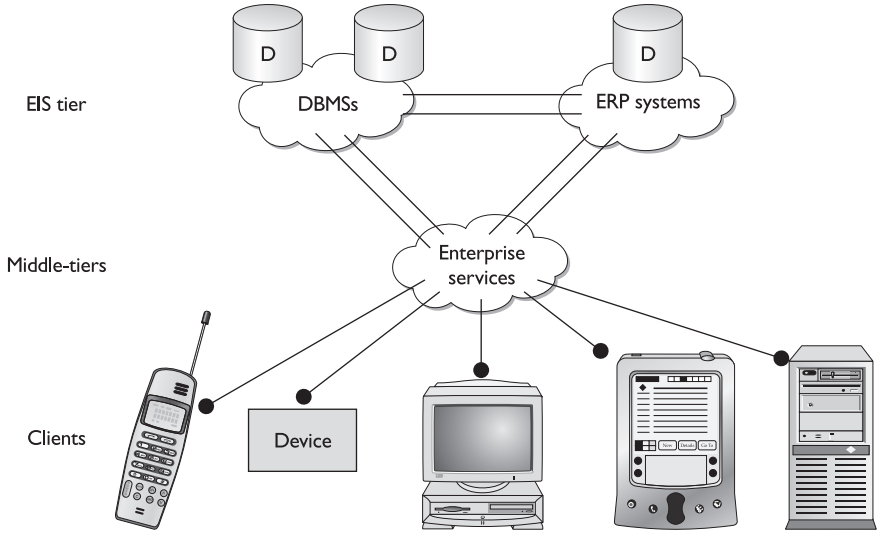
As more businesses move toward an e-business strategy, integration with existing EISs becomes the key to success. Enterprises with successful e-businesses need to integrate their existing EISs with new web-based applications, possibly using the JEE APIs that match the existing EIS functionality. JEE APIs extend the reach of the EISs to support business-to-business (B2B) transactions.

Before the JEE Connector architecture was defined, no specification for the Java platform addressed the problem of providing a standard architecture for integrating heterogeneous EISs. Most EIS vendors and application server vendors use nonstandard vendor-specific architectures to provide connectivity between application servers and EISs. Figure 4-14 illustrates the complexity of a heterogeneous environment.

The JEE Connector architecture provides a Java solution to the problem of connectivity among the many application servers and EISs already in existence. By using the JEE Connector architecture, EIS vendors no longer need to customize their products for each application server. Application server vendors who conform to the JEE Connector architecture do not need to add custom code whenever they want to add connectivity to a new EIS.

FIGURE 4-14

The heterogeneous enterprise architecture



The JEE Connector architecture is based on the technologies that are defined and standardized as part of the JEE.

JEE Connector Overview

The JEE Connector architecture defines a standard architecture for connecting the JEE platform to heterogeneous EISs. Examples of EISs exist in almost any enterprise computing environment. A nonexhaustive list includes ERP, mainframe transaction processing, database systems, and legacy applications not written in the Java programming language. By defining a set of scalable, secure, and transactional mechanisms, the JEE Connector architecture enables the integration of EISs with application servers and enterprise applications.

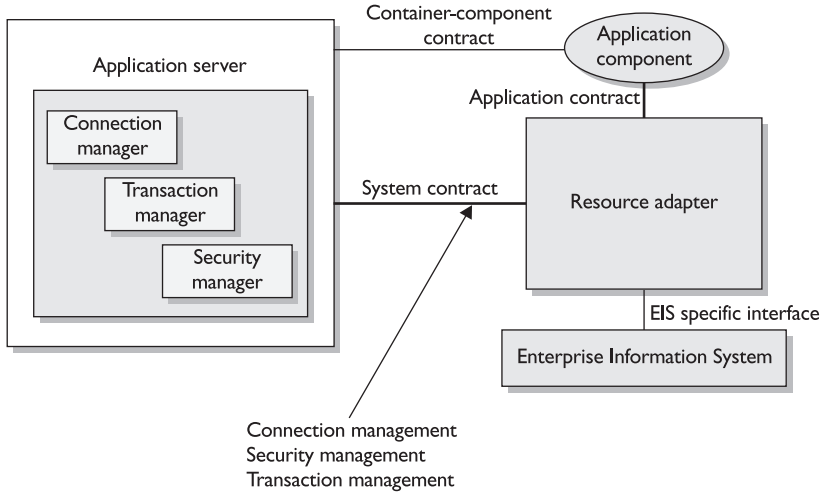
The JEE Connector architecture enables an EIS vendor to provide a standard resource adapter for its EIS. The resource adapter plugs into an application server, providing connectivity among the EIS, the application server, and the enterprise application. If an application server vendor has extended its system to support the JEE Connector architecture, it has connectivity to multiple EISs. An EIS vendor needs to provide just one standard resource adapter that has the capability to plug into any application server that supports the JEE Connector architecture.

Multiple resource adapters (that is, one resource adapter per type of EIS) are pluggable into an application server. This capability enables application components deployed on the application server to access the underlying EISs.

Figure 4-15 illustrates the JEE Connector architecture.

FIGURE 4-15

The JEE Connector architecture



Resource Adapter

To achieve standard, system-level pluggability between application servers and EISs, the JEE Connector architecture defines a standard set of system-level contracts between an application server and EIS. The resource adapter implements the EIS side of these system-level contracts.

A *resource adapter* is a system-level software driver used by an application server or an application client to connect to an EIS. By plugging into an application server, the resource adapter collaborates with the server to provide the underlying mechanisms, the transactions, security, and connection pooling mechanisms. A resource adapter is used within the address space of the application server.

System Contract

An application server and an EIS collaborate to keep all system-level mechanisms, such as transactions, security, and connection management, transparent from the application components. As a result, an application component provider focuses on the development of business and presentation logic for its application components and need not get involved in the system-level issues related to EIS integration. This promotes easier and faster development of scalable, secure, and transactional enterprise applications that require connectivity with multiple EISs.

The JEE Connector architecture defines the following set of system-level contracts between an application server and EIS:

- A Connection Management contract that lets an application server pool connect to an underlying EIS and lets application components connect to an EIS. This leads to a scalable application environment that can support a large number of clients requiring access to EISs.
- A Transaction Management contract between the transaction manager and an EIS that supports transactional access to EIS resource managers. This contract lets an application server use a transaction manager to manage transactions across multiple resource managers. This contract also supports transactions that are managed internal to an EIS resource manager without the necessity of involving an external transaction manager.
- A Security Contract that enables a secure access to an EIS. This contract provides support for a secure application environment, which reduces security threats to the EIS and protects valuable information resources managed by the EIS.

Common Client Interface (CCI)

The JEE Connector architecture also defines a common client interface (CCI) for EIS access. The CCI defines a standard client API for application components. The CCI enables application components and enterprise application integration (EAI) frameworks to drive interactions across heterogeneous EISs using a common client API. The CCI is intended for use by the EAI and enterprise tools vendors.

CERTIFICATION OBJECTIVE 4.11

Illustrate When to Use JEE Technology for Given Situations

This scenario illustrates the use of the JEE Connector architecture in a B2B e-commerce scenario. Morris Phillips Corp. is a manufacturing firm that aims to adopt an e-business strategy. Morris Phillips has huge existing investments in its EIS systems, which include an ERP system and a mainframe transaction processing system. Morris Phillips needs to drive B2B interactions with its multiple supplier vendors, and it wants to leverage its existing EIS investment while adopting the new e-business architecture.

With these goals in mind, Morris Phillips buys a JEE-based server (called the B2B server) from B2B, Inc. The B2B server can drive B2B interactions with multiple buyers and suppliers. The B2B interactions are driven using XML over HTTP (or HTTPS). The JEE Connector architecture enables Morris Phillips to integrate its existing EISs with the B2B server. Morris Phillips buys off-the-shelf resource adapters for its existing set of EISs. It then integrates its B2B server and applications (deployed on the B2B server) with its EISs using these connectors.

The applications deployed on the B2B server extract data from the underlying EISs. The extracted data may be in XML or converted into XML by the applications. The loosely coupled B2B interactions with suppliers are then driven by exchanging XML data over the HTTP (or HTTPS) protocol.

Figure 4-16 illustrates Morris Phillips' use of the JEE Connector architecture to run its business.

EISs provide the information infrastructure critical to the business processes of an enterprise. Examples of EISs include relational databases, ERP systems, mainframe transaction processing systems, and legacy database systems.

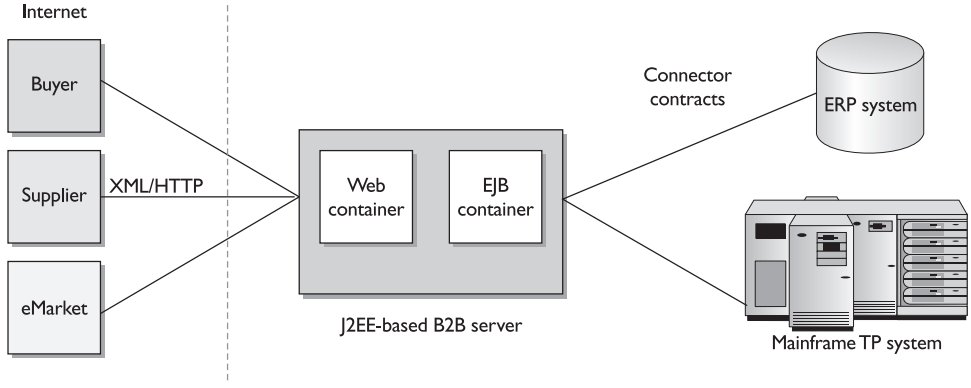


FIGURE 4-16

Morris Phillips' use of the JEE Connector architecture

The EIS integration problem has assumed great importance because enterprises are striving to leverage their existing systems and resources while adopting and developing new technologies and architectures. Today, enterprise application development is more about integration rather than developing an enterprise application from scratch. Enterprises cannot afford to discard their investments in existing applications and EISs. The emergence of web-based architectures and Web services has made it more imperative for enterprises to integrate their EISs and applications and expose them to the web.

The EIS integration problem is one part of the broader scope of EAI. EAI entails integrating applications and enterprise data sources so that they can easily share business processes and data. We will focus on the following aspects of EAI, including discussions of recommended guidelines:

- **Application integration** Existing enterprise applications may be off-the-shelf bundled applications, or they may be developed in-house. Two examples are supply chain management (SCM) and customer relationship management (CRM) applications. While such applications expose business-level functionality used directly by end users or integrated with other enterprise applications, they usually do not expose the underlying data on which the business functionality is built.
- **Data integration** An enterprise environment often contains more than one database system upon which its business processes run. These database systems may be relational, object-based, hierarchical, file-based, or legacy stores. Data integration focuses on integrating existing data with enterprise applications. For example, an integration might entail integrating a web-based order management system with an existing order and customer database.

- **Legacy integration** Legacy integration involves integrating new enterprise applications with applications and EISs that have been in operation for some time, often referred to as an enterprise's *legacy* systems. An enterprise cannot afford any disruption in these legacy systems.

EIS Guidelines: Data Access

Here, the architect must rely on vendor tools for EIS integration, such as data and function mining tools, object-oriented analysis and design tools, application code generation tools, application composition tools, and of course deployment tools. In-house deployers who are knowledgeable in the enterprise organization should be charged to set transaction, security, and deployment requirements.

EIS Access Objects

Access objects abstract complex, low-level details of EIS system access into access objects; provide a common, consistent access to various types of EISs; and separate access objects from business objects. Access objects can be made into well-known JavaBeans for use in development tools.

When implementing access objects, do not make assumptions about environments outside access objects. Architects should strive to design for reusability across tiers and components. Access objects should not define declarative transactions or security requirements. It is important to maintain consistency in programming restrictions between business objects and access objects.

Guidelines: Connections

Components should acquire and release connections within a single method. They should account for differences across component types in connection management such as JSP and servlets, stateful and stateless session beans, and entity beans. Components should avoid opening multiple concurrent connections to a single database, because this is not supported by some JDBC drivers.

Performance-Related Best Practices: Data Caching

Much of the interesting data in a system can remain unchanged for minutes, hours, or even longer. When accessing interesting information of such a static nature, or in noncrucial use cases wherein a client does not require real-time data, network traffic

and database usage can be greatly reduced through the use of data caching. Here are some tips:

- If clients have slow back-end connections, consider compressing data for network communication.
- Minimize the number of network round trips required by the application.
- For applications to scale to many users, minimize the amount of shared memory that requires updating.
- Cache data to minimize lookup time, although this can reduce scalability if locks are required to access the cache.
- If there are more accesses than updates to a cache, share the access lock among all the accessors, but be aware that this reduces the window for updaters to lock the cache.
- Eliminate memory leaks before tuning execution speed.
- Ensure that the development environment approximates/simulates the production deployment environment.
- Consider metrics: maximum response ratio and CPU utilization under various activity loads. How well does the application scale when additional users are added?
- Spend your time wisely: For example, improving the performance of a method that is called 1000 times is better than improving the performance of a method that is called only 10 times.
- Don't cache data unless you know how and when to invalidate the cached entries.

JEE Best Practices: Services

The Service-locator is used when accessing common services within JEE, such as JMS factories, data sources, *EJBHome* objects, and so on. The application will use the JNDI API. This will require the creation of an initial context object (the root of the naming service), followed by a lookup or search for the desired resource or service. This lookup (if successful) results in the transferring of that resource's serializable representative to the interested party.

Some major design problems become evident when the clients are co-located and not local to the EJB or other accessed service:

- Entire seconds can go by each time these operations are carried out.
- Each interested client must be complicated by the inclusion of JNDI-API-specific code.
- Unnecessary network traffic is added to the system.

Avoid some of this overhead by caching references to the service object (for example, *EJBHome* references) to avoid JNDI lookup overhead. An EJB call is expensive, owing to the requirements for an EJB. For example, a method call from the client could cover all the following: get Home reference from the NamingService (one network round trip); get EJB reference (one or two network round trips plus remote creation and initialization of Home and *EJBObjects*); call method and return value on *EJBObject* (two or more network round trips: client/server and [multiple] server-DB; several costly services used such as transactions, persistence, security, and so on; multiple serializations and deserializations).

To prevent performance problems, do the following:

- If an EJB service for an object is overkill (for example, if the object will only be accessed locally), use a plain Java object and not an *EJBObject*.
- You can use local interfaces (from EJB2.0) if EJB calls another EJB (or a servlet calling an EJB) from the same container and the same virtual machine.
- Wrap entity beans in a session bean to change multiple EJB remote calls into one session bean remote call and several local calls. (This is the pattern discussed earlier in the chapter, known as the *session bean facade*.)
- Change multiple remote method calls into one remote method call with all the data combined into a parameter object.
- Control serialization by modifying unnecessary data variables with a transient keyword to avoid unnecessary data transfer over network.
- Cache *EJBHome* references to avoid JNDI lookup overhead (as we just discussed the pattern called *ServiceLocator*).
- Declare nontransactional methods of session beans with *NotSupported* or *Never* transaction attributes (in the *ejb-jar.xml* deployment descriptor file). If the code calling a method in this bean has a transaction running, suspend that transaction until the method called in this bean completes. No transaction context is created for this bean.
- Transactions should span the minimum time possible as transactions lock database rows. This depends on transaction isolation that is defined in terms of isolation levels called dirty reads, repeatable reads, and phantom reads.

A repeatable read is when the data read is guaranteed to look the same if read again during the same transaction. Repeatable reads typically means that the data read is locked against changes. If the data is locked, it cannot be changed by any other transaction until this transaction ends. A dirty read occurs when the first transaction reads uncommitted changes made by a second transaction. If the second transaction is rolled back, the data read by the first transaction becomes invalid because the rollback undoes the changes. The first transaction won't be aware that the data it has read has become invalid. Phantom reads occur when new rows added to the database are detectable by transactions that started prior to the insert. Queries will include rows inserted by other transactions after their transaction has started.

Security

Threats to enterprise-critical assets can include such events as disclosure of confidential information, the modification or destruction of information, the misappropriation of protected resources, the compromise of accountability, and misappropriation that compromises availability. Exposure to threats can be mitigated using software that provides authentication, authorization, and auditing.

A good security guideline is to support consistent end-to-end security architecture. This is accomplished by integrating with existing security environments. This is known as *identity management*. Large firms have provisioning systems that manage user accounts across different application domains and operating systems. The proper security guidelines should support authentication and authorization. Another objective of good security is to be transparent to application components and enable applications to be portable across security environments. A good technique is to have the user identity passed from the calling application.

Security Terminology

An *entity* is something that can have access rights applied to it. A *principal* is an entity to which privileges can be assigned. A *role* is a collection of privileges. *Authentication* is a mechanism by which callers and service providers prove that they

are acting on behalf of specific users or systems. Web-tier authentication consists of HTTP basic authentication, form-based authentication, and HTTPS mutual authentication.

Authentication in the EJB and EIS Integration Tiers

The EJB tier authentication can be accomplished using protection domains, by placing a protected web resource in front of a protected EJB resource, or by linking protected web resources on every web resource that calls EJB resources. On the EIS integration tier, authentication can be accomplished using container-managed resource manager sign-on or an application-managed resource manager sign-on.

Protection Domains

In a JEE application, *protection domain* refers to a set of entities that are *assumed*, or known to trust each other. When a component interacts with components in the same protection domain, no constraint is placed on the identity that it associates with its call. The caller may *propagate* the caller's identity, or choose an identity based on knowledge of authorization constraints imposed by the called component, since the caller's ability to claim an identity is based on trust. If the concept of protection domains is employed to avoid the need for authentication, there must be a means to establish the boundaries of protection domains, so that trust in unproven identities does not cross these boundaries.

In JEE, a container provides an authentication boundary between external callers and the components it hosts. Containers enforce the boundaries, and implementations are likely to support protection domains that span containers. A container is not required to host components from different protection domains, although an implementation may choose to do so.

Authorization

Authorization entails applying security policies to regulate what specific users, or groups of users, can access in the system. An access control limits the resources a user has access to according to that user's permissions. Access control can also be used to limit the type of access a user has to a resource, such as read or write access. Two approaches can be used to define access control rules: the capabilities are examined to focus on what a caller can do, and permissions focus on who can do what. The JEE application programming model focuses on permissions.

With declarative authorization, the container-enforced access control rules associated with a JEE application are established by the deployer. The deployer uses a deployment tool to map an application permission model to policy specific to the operational environment. The application permission model is contained in a deployment descriptor.

The deployment descriptor defines logical privileges, called *security roles*, and associates them with components to define the privileges required to be granted permission to access components. The deployer assigns these logical privileges to specific callers to establish the capabilities of users in the runtime environment. Callers are assigned logical privileges based on the values of their security attributes.

The EJB container grants permission to access a method to callers that have at least one of the privileges associated with the method. Security roles also protect web resource collections—that is, a URL pattern and an associated HTTP method, such as `GET`. The web container enforces authorization requirements similar to those for an EJB container. Note that when a resource has no associated security role, permission to access the resource will be granted to all.

In both tiers, access control policy is defined at deployment time, rather than application development. The deployer can modify the policy provided by the application assembler. The deployer refines the privileges required to access the components and defines the correspondence between the security attributes presented by callers and the container privileges. In any container, the mapping from security attributes to privileges is scoped to the application, so that the mapping applied to the components of one application may be different from that of another application.

With programmatic authorization, a JEE container makes access control decisions before dispatching method calls to a component. As a result, the state of a component doesn't affect the access decisions. A component can use two methods, `EJBContext.isCallerInRole` for enterprise bean code and `HttpServletRequest.isUserInRole` for web components. A component uses these methods to determine whether a caller has been granted a privilege selected by the component, basing its choice on the parameters, the state of the component, or factors such as the time of the call.

The application component provider of a component that calls one of these functions must declare the complete set of distinct *`roleName`* values used in all of its calls. These declarations appear in the deployment descriptor as *`security-role-ref`* elements. Each *`security-role-ref`* element links a privilege name embedded in the application as a *`roleName`* to a security role. It is ultimately the deployer that establishes the link between the privilege names embedded in the

application and the security roles defined in the deployment descriptor. The link between privilege names and security roles may differ for components in the same application.

Use declarative authorization where possible and programmatic authorization when more functionality is required. When using declarative authorization, ensure that access control is not bypassed. Apply the same access control rules to all the methods in a component. There is a trade-off between the external access control policy configured by the deployer and the internal policy embedded in the application code. The former is flexible after the application has been written. The latter provides more options in terms of functionality. The former is transparent and completely comprehensible. The latter is hidden in the application code and may be understood only by the application developers. These trade-offs should be considered in choosing the authorization model.

Controlling Access to Resources

To control access to web resources, specify the constraint in the deployment descriptor. To control access to EJB resources, specify the roles in the deployment descriptor. You can also specify the methods of the remote and home interface that each security role is allowed to invoke. The proper assignment of users to roles determines whether a resource is protected.

To ensure message integrity, the following measures can be used:

- **Message signature** A cryptographically enciphered message digest of the message contents
- **Message confounder** Ensures message authentication is useful only once

Message signatures might be required for component-to-component invocations that traverse unprotected networks. Specify message protection only for critical messages and components in the deployment descriptor.

Transactions

A *transaction* is a bracket of processing that represents a logical unit of work; it is an "all-or-nothing" contract, and all of the processing must be completed or else the transaction management should restore the application to the status quo ante—as it was before the transaction. Transactions are basically a specific sequence of operations on resources, typically the data actions *select*, *insert*, and *update*, which

transform the system from one consistent state to another. To reflect the correct state of the system, a transaction should have the following properties:

- **Atomicity** This is the all-or-nothing property. Either the entire sequence of operations is successful or the sequence is entirely unsuccessful. Completed transactions are committed. Partially executed transactions are rolled back.
- **Consistency** A transaction maps one consistent state of the resources to another. Consistency is concerned with correctly reflecting the reality of the state of the resources.
- **Isolation** A transaction should not reveal its results to other concurrent transactions before it commits. Certain isolation levels (serialization) assure that transactions do not access data that is being concurrently updated.
- **Durability** The results of the committed transactions are permanent. Resource managers ensure that the results of a transaction are not altered due to system failures.

Transactions ensure data integrity by controlling access to data. This frees an application programmer from the complex issues of failure recovery and multiple-user programming. Transactions are a mechanism used for simplifying the development of distributed multiuser enterprise applications. Two types of transaction demarcation can be used: bean-managed and container-managed. In container-managed transaction demarcation, six different transaction attributes—`Required`, `RequiresNew`, `NotSupported`, `Supports`, `Mandatory`, and `Never`—can be associated with an enterprise bean's method.

Transaction Guidelines in the Web Tier

A servlet or JSP can use JNDI to look up a *UserTransaction* and use the Java Transaction API (JTA) to demarcate transactions. A servlet should start a transaction only in its service method. A transaction should not span multiple web requests. It is typically bracketed by a *begin* and a *commit* or *rollback*, as the following code snippet illustrates.

```
Context ic = new InitialContext();UserTransaction
t =(UserTransaction)ic.lookup("java:comp/UserTransaction");t.begin();
// perform processingif (everything_worked){t.commit();}else{t.rollback();}
```

In a multi-tier environment, when using EJB, the use of JTA in the web tier is not recommended. In bean-managed transaction demarcation, the EJB bean uses *UserTransaction*. Only session beans can choose to use bean-managed transactions.

In container-managed transaction demarcation, the EJB container is responsible for transaction demarcation. Moreover, you should use container-managed transaction demarcation because it is less prone to error, and you should let the container handle transaction demarcation automatically. It frees the component provider from writing transaction demarcation code in the component. It is easier to group enterprise beans to perform a certain task with specific transaction behavior. The bottom line is that the application assembler can customize the transaction attributes in the deployment descriptor without modifying the code.

EJB Tier—Container-Managed

Transaction *demarcation* is the vehicle by which transaction behavior of EJB is specified declaratively; it frees the developer from writing code. It is less error-prone because the container handles all the transaction servicing. It is easier to compose multiple enterprise beans to perform a certain task with transaction behavior. It can result in improved performance. A transaction attribute supports declarative transaction demarcation and conveys to the container the intended transactional behavior of the associated EJB component's method.

Six transactional attributes are possible for container-managed transaction demarcation:

- **NotSupported** The bean runs outside the context of a transaction. Existing transactions are suspended during method calls. The bean cannot be invoked within a transaction. An existing transaction is suspended until the method called in this bean completes.
- **Required** Method calls require a transaction context. If one already exists, it will be used; if one does not exist, it will be created. The container starts a new transaction if no transaction exists. If a transaction exists, the bean uses that transaction.
- **Supports** Method calls use the current transaction context if one exists but don't create one if none exists. The container will not start a new transaction. If a transaction already exists, the bean will be included in that transaction. Note that with this attribute, the bean can run without a transaction.
- **RequiresNew** Containers create new transactions before each method call on the bean and commit transactions before returning. A new transaction is always started when the bean method is called. If a transaction already exists, that transaction is suspended until the new transaction completes.

- **Mandatory** Method calls require a transaction context. If one does not exist, an exception is thrown. An active transaction must already exist. If no transaction exists, the *javax.ejb.TransactionRequired* exception is thrown.
- **Never** Method calls require that no transaction context be present. If one exists, an exception is thrown. The bean must never run with a transaction. If a transaction exists, the *java.rmi.RemoteException* exception is thrown.

With respect to transaction attributes, you should use *Required* for the default transaction attribute. The *RequiresNew* attribute is useful when the bean methods need to commit unconditionally. The *NotSupported* attribute can be used when the resource manager is not supported by the JEE product. The BluePrint recommends not using the attribute *Supports*. *Mandatory* and *Never* can be used when it is necessary to verify the transaction is associated with the client.

Transaction Guidelines in EIS

For proper handling of transactions within the EIS integration tier, it is recommended that a component uses JTA whenever possible when accessing EIS systems. Using JTA transaction allows multiple components accessing EIS to be grouped in a single transaction. If a component marks the transaction as rollback only, all EIS work will be rolled back automatically. With local transactions, each EIS accessed will have to be committed or rolled back explicitly. In addition, components need extra logic to deal with individual EIS rollbacks or failures.

To handle a group of EIS operations to work as a transaction, you might need compensating transactions. For example, in an "identity management system," let's say that when a new user starts his job, EJBs are used to create a Windows NT ID, a UNIX ID, and a mainframe ID. We want this group of provisioned applications to be a transaction. Suppose the NT and UNIX IDs are created but the mainframe fails; we need to compensate or undo the transactions for the NT and UNIX IDs.

A compensating transaction is a *transaction*, or group of operations, used to undo the effect of a previously committed transaction. They are useful if a component needs to access an EIS that does not support JTA.

A number of problems can arise when using compensating transactions: It is not always possible to undo the effect of a committed transaction, and the required atomicity could be broken if the server crashes when a compensating transaction is used. In addition, database "locks" notwithstanding, inconsistent data might be seen by concurrent EIS access.

EXERCISE 4-10

Security Guidelines

Question Describe security guidelines, terminology, and forms of authentication.

Answer A good security guideline is to provide a consistent end-to-end security architecture. This is accomplished by seamlessly integrating with existing security environments such as EIS support authentication and authorization. Another objective of good security is to be transparent to application components and enable applications to be portable across security environments.

With respect to security terminology, an *entity* is something that can have access rights applied to it. A *principal* is an entity to which privileges can be assigned. A *role* is a collection of privileges. An *authentication mechanism* is one by which callers and service providers prove that they are acting on behalf of specific users or systems. Good web-tier authentication can consist of HTTP basic authentication, form-based authentication, and HTTPS mutual authentication for transactions that need added security for sensitive data.

EXERCISE 4-11

The Role of Transactions

Question Describe the role of transactions.

Answer Transactions ensure data integrity by controlling access to data. Transactions free an application programmer from the complex issues of failure recovery and multiple-user programming. Transactions are a mechanism for simplifying the development of distributed multiuser enterprise applications. Transactions span across all tiers.

CERTIFICATION SUMMARY

As you have seen, the JEE platform is a multi-tiered distributed application model, where application logic is divided into components according to their function. The various components of a JEE application are installed on different machines. A component's location depends on which tier or layer in the multi-tiered JEE environment that component belongs to. These components will already exist (legacy, client/server databases, messaging) and must be integrated with the JEE components. The enterprise architect must be aware of the way in which the JEE application framework can be used to integrate seamlessly with the existing myriad of business components that make up the enterprise environment.

As you have seen in the chapter, these components reside at various tiers in the framework. The architect must understand the client tier components, web tier components, and business tier components that run on the JEE server, and, probably most important for the enterprise, the EIS tier.

TWO-MINUTE DRILL

Explain the JEE Architecture and System Requirements

- ❑ While a JEE application can consist of three or more tiers or layers, JEE multi-tiered applications are generally considered to be three-tiered applications because they are distributed across three different locations: client machines, JEE server machine, and the database or legacy machines at the back end. JEE applications consist of client components, web components, and business components.
- ❑ JEE applications are made up of components: self-contained functional software units assembled into JEE applications with their related classes and files. These components communicate with other components.
- ❑ The component-based and platform-independent JEE architecture facilitates development, because business logic is organized into reusable components, and the JEE server provides underlying services in the form of a container for every component type.
- ❑ A JEE application is usually assembled from two different types of modules: enterprise beans and web components. Both of these modules are reusable; therefore, new applications can be built from pre-existing enterprise beans and components. The modules are also portable, so the application that comprises them will be able to run on any JEE server conforming to the specifications.

Explain the Use of Patterns in the JEE Framework

- ❑ The JEE framework employs design patterns to support these capabilities. JEE uses the following core patterns to enable flexible association of EJB classes with other components. The Proxy pattern provides a separate implementation of interface and working code for location transparency. The Decorator provides a similar contract for a class but with added functionality. The Factory Method provides ability to define a contract for creating an object but defers instantiation to subclasses. The Abstract Factory provides a contract for creating families of related or dependent objects without specifying concrete classes.
- ❑ The use of best practices, design patterns, and guidelines is important for JEE architects. Successful architects and developers share their knowledge and pass on their proven techniques to others. The net result is productivity. The ultimate product is the implementation of solid applications.

Describe the Concepts of "Best Practices" and "Guidelines"

- ❑ A best practice is an optimal process that is recognized and approved by peers in similar situations. It is applicable to a cross-section of scenarios with varying resources and sizes. It takes design requirements into consideration.
- ❑ A guideline is a rule applied horizontally to the design. Guidelines reflect agreements on practices or operations by recognized professional associations. This includes formal, approved standards, as contrasted to de facto standards and proprietary standards that are exceptions to this concept.

Illustrate the Use of JEE for Workflow

- ❑ A common method for designing applications is to organize them around an event-driven user interface. Utilizing the MVC design pattern best practice results in a separation of the application data from the ways that the data can be accessed or viewed as well as from the mapping between system events (such as user interface events) and application behaviors.

Review Best Practices Applicable for All Tiers

- ❑ The Enterprise JavaBeans (EJB) tier hosts the application-specific business objects and the system-level services (such as transaction management, concurrency control, and security). The EJB tier is a critical link between the web tier and the EIS integration tier. It typically hosts the entity beans and session beans, data access objects and value objects, and perhaps master-detail modeling using enterprise beans.

Review Best Practices for the Client Tier

- ❑ Thin-client solutions (HTML on a browser) are important to Internet-based applications. The browser acts as your client for rendering the presentation as encoded in HTML.
- ❑ In addition to what can be rendered with static HTML, the following items can be used to create web content: JSPs, servlets, applets, and JavaScript can be used to enhance the browser interface.

Enumerate the Components and Categories of the Web Tier

- ❑ The two types of components currently specified for the web tier are servlets and JSP pages.

- ❑ Web components are hosted by servlet containers, JSP containers, and web containers.
- ❑ In addition to standard container services, a servlet container provides network services by which requests and responses are sent and that decode requests and format responses. All servlet containers must support HTTP as a protocol for requests and responses, but they may also support additional request-response protocols such as HTTPS.
- ❑ A JSP container provides the same services as a servlet container and an engine that interprets and processes a JSP page into a servlet.
- ❑ A web container provides the same services as a JSP container and provides access to the JEE service and communication APIs.

Explain How to Apply MVC to the Web Tier

- ❑ MVC is applied to the web tier by separating the application data from the ways that the data is accessed or viewed. The MVC pattern consists of three component types:
- ❑ The Model, usually a JavaBean or an EJB, represents the application data along with methods that operate on that data.
- ❑ The View component, usually a JSP, displays the data to the user.
- ❑ The Controller, which is usually a servlet, translates user actions such as mouse movement and keyboard input and dispatches operations on the Model.

Review the Best Practices for the Presentation Layer

- ❑ Separate HTML from Java.
- ❑ Try to place business logic in JavaBeans.
- ❑ Factor general behavior out of custom tag handler classes.
- ❑ Favor HTML in Java handler classes over Java in JSPs.
- ❑ Use an appropriate inclusion mechanism.
- ❑ Use a JSP template mechanism.
- ❑ Use style sheets.
- ❑ Use the MVC pattern.
- ❑ Use available custom tag libraries.
- ❑ Determine the appropriate level of XML compliance.
- ❑ Use JSP comments in most cases.

- ❑ Follow HTML best practices.
- ❑ Utilize the JSP exception mechanism.

Review the Internationalization and Localization

- ❑ The set of political, cultural, and region-specific elements represented in an application is called a *locale*. Applications should customize data presentation to each user's locale. Internationalization, also known as *I18n*, is the process of separating locale dependencies from an application's source code. Examples of locale dependencies include messages and user interface labels, character sets, encoding, and currency and time formats. Localization (also called *L10n*) is the process of adapting an internationalized application to a specific locale. An application must first be internationalized before it can be localized. Internationalization and localization make a JEE application available to a global audience.

Illustrate When to Use JEE Technology for Given Situations

- ❑ With respect to security, an entity is something that can have access rights applied to it. A principal is an entity to which privileges can be assigned. A role is a collection of privileges.
- ❑ Authentication is a mechanism by which callers and service providers prove that they are acting on behalf of specific users or systems. Web-tier authentication consists of HTTP basic authentication, form-based authentication, and HTTPS mutual authentication.
- ❑ Authorization entails applying security policies to regulate what specific users, or groups of users, can access in the system. An access control limits the resources a user can access based on permissions. Access control can also be used to limit the type of access a user has to a resource, such as read or write access. There are two approaches to defining access control rules: capabilities are examined to focus on what a caller can do, and permissions focus on who can do what.
- ❑ For proper handling of transactions within the EIS integration tier, it is recommended that a component uses JTA whenever possible when accessing EIS systems. Using JTA transaction allows multiple components accessing EIS to be grouped in a single transaction. If a component marks the transaction as rollback only, all EIS work will be rolled back automatically.

SELF TEST

The following questions will help you measure your understanding of the material presented in this chapter. Read all the choices carefully because there might be more than one correct answer. Choose all correct answers for each question.

Explain the JEE Architecture and System Requirements

1. Which of the following is not true about JEE containers?
   - A. An EJB container manages the execution of all enterprise beans for a single JEE application. Enterprise beans and their accompanying containers run on the JEE server.
   - B. A web container manages the execution of all JSP and servlet components for a single JEE application. Web components and their accompanying container run on the JEE server.
   - C. An application client container manages the execution of all application client components for a single JEE application. Application clients and their accompanying containers run on the JEE server.
   - D. An applet container is the web browser and Java plug-in combination that runs on the client machine.
2. Which statement is *not* true when discussing the EJB tier?
   - A. The Enterprise JavaBeans (EJB) tier hosts the application-specific business objects.
   - B. The Enterprise JavaBeans (EJB) tier does not host system-level services (such as transaction management, concurrency control, and security); they are hosted on the EIS tier.
   - C. The EJB tier is a link between the web tier and the EIS integration tier.
   - D. The EJB tier hosts the entity beans and session beans, data access objects and value objects, and perhaps master-detail modeling using enterprise beans.
3. Which of the following is *not* true when put in the context of JEE transaction processing?
   - A. A compensating transaction is a transaction, or group of operations, used to undo the effect of a previously committed transaction.
   - B. When choosing a transaction attribute, use *Required* for the default transaction attribute.
   - C. When choosing a transaction attribute, use *RequiresNew* when the bean methods need to commit unconditionally.
   - D. When using a compensating transaction, it is always possible to undo the effect of a committed transaction.

Explain the Use of Patterns in the JEE Framework

4. JEE uses the core patterns to enable flexible association of EJB classes with other components. Which of the following is *not* used by JEE?

A. Proxy
B. Decorator
C. Designer
D. Factory

Describe the Concepts of "Best Practices" and "Guidelines"

5. Which statement is *not* true when discussing best practices?

A. Data access objects are a useful best practice, as they encapsulate access to data and maintain a clean separation of bean and database access code.
B. A session bean facade provides a simple, single point of entry to shared entity beans.
C. A session bean facade does not shield the client from complex entity bean relationships and manages workflow on a client's behalf.
D. A session bean facade avoids the problems associated with access to entity beans from the client layer—namely overabundance of network traffic and latency and awkward security management.

Review Best Practices Applicable for All Tiers

6. In which of the following cases would an application not necessarily benefit from the use of Enterprise JavaBeans?

A. Small-scale deployment
B. Large-scale deployment
C. Requirements transactional in nature
D. No transactional requirements

7. The JEE platform uses a multi-tiered distributed application model; which of the following is not considered a tier in this architecture?

A. Client tier
B. Web tier
C. Enterprise information system (EIS) tier
D. Security tier

8. Which of the following are "best practices" for large distributed systems?
   A. Avoid business logic implementation in the display code; display inconsistencies can result because the logic can be copied and modified in one object and not another.
   B. Coding data manipulation logic, format and display code, and user event handling together can make application maintenance simple.
   C. Facilitate reuse of user interfaces by segregating application logic from the code for an existing interface.
   D. Utilizing the MVC design pattern results in a separation of the application data from the ways that the data can be accessed or viewed as well as from the mapping between system events (such as user interface events) and application behaviors.
9. Which of the following are *not* benefits of using the MVC best practice?
   A. Clarifies application design through separation of data modeling issues from data display and user interaction
   B. Enhances reusability by separating application functionality from presentation
   C. Facilitates maintenance by encapsulating application functions behind trusted APIs
   D. Simplifies database design because only the View components access the database
10. Which of the following is *not* true of using the MVC best practice?
   A. The Model in an MVC-based application can be divided into two parts: the internal state of the system and the actions that can be taken to alter that state.
   B. The Controller portion of the application focuses on receiving requests from the client (most often a user running a web browser), deciding what business logic function is to be performed, and delegating responsibility for producing the next phase of the user interface to an appropriate View component.
   C. The Model determines how the results should be displayed.
   D. The View transfers user input to the Controller.

Explain How to Apply MVC to the Web Tier

11. Which of the following are not components of the MVC?
   A. Model
   B. Calculator
   C. View
   D. Controller
12. Which of the following is *not true* of the MVC?
   A. The View extrapolates data presentation and responds to users with data.
   B. The Controller extrapolates the user interaction/application semantic map and transforms user actions into application actions.

C. The Model manages persistence.
D. The Controller maintains the application state.

Review the Best Practices for the Presentation Layer

13. Which of the following is *not* typically considered a threat to enterprise-critical assets?
A. The disclosure of confidential information
B. The modification or destruction of information
C. The misappropriation of protected resources
D. A misappropriation that does not compromise availability

Illustrate When to Use JEE Technology for Given Situations

14. Which of the following are not true about screen scrapers?
A. Screen scrapers function as terminal emulators on one end and as object interfaces on the other.
B. Screen scraping may be a useful tool when used in conjunction with the off-board servers.
C. Changes to legacy UI have little or no impact on the new GUI.
D. Screen scraping is best used when the legacy clients have loose coupling with other tiers.

15. If the telephone company were to rewrite its existing legacy code using newer JEE technology, what technology would you choose to accommodate both the block purchase and the individual query?
A. Java Applet technology for the CORBA call and custom socket programming for vanity number requests
B. Java Servlet API for the CORBA call and JSP for the custom socket programming
C. Entity EJBs for both
D. Session EJBs for both
E. JNDI for both
F. MQ Series with a JMS-based solution for both

16. Your company's web site offers the customers price comparisons on a variety of different products. You are in charge of converting the web-based solution over to the appropriate JEE technology. Which of the following should you use?
A. JSP, servlets
B. JSP, servlets, EJBs
C. Applets, EJBs
D. No need to change it
E. Perl/CGI scripts is the best solution

17. Regarding the JEE EIS integration, which of the following statements is *not* true?

A. Before the JEE Connector architecture was defined, no specification for the Java platform addressed the problem of providing a standard architecture for integrating heterogeneous EISs.

B. The JEE Connector architecture provides a Java solution to the problem of connectivity between the many application servers and only new EISs, not those already in existence.

C. Application server vendors who conform to the JEE Connector architecture do not need to add custom code whenever they want to add connectivity to a new EIS.

18. Regarding the JEE EIS integration contracts, which of the following statements is *not* true?

A. A Connection Management contract allows an application server to pool connections to an underlying EIS.

B. A Transaction Management contract lets an application server use a transaction manager to manage transactions across multiple resource managers.

C. A Security Contract provides support for a secure application environment, which reduces security threats to the EIS and protects valuable information resources managed by the EIS.

D. A Transaction Management contract does not support transactions that are managed internal to an EIS resource manager without the necessity of involving an external transaction manager.

19. Which is the following is *not* true about enterprise applications and integration?

A. Data integration focuses on integrating existing data with enterprise applications. For example, an integration might entail integrating a web-based order management system with an existing order and customer database.

B. Legacy integration involves integrating new enterprise applications with applications and EISs that have been in operation for some time, often referred to as an enterprise's *legacy* systems.

C. Application integration occurs when existing enterprise applications may be off-the-shelf bundled applications or they may be developed in-house.

D. Enterprise application development is about building an enterprise application from scratch.

SELF TEST ANSWERS

Explain the JEE Architecture and System Requirements

1. ☑ **C** is correct. An application client container manages the execution of all application client components for a single JEE application. Application clients and their accompanying container run on the client's machine and not the JEE server.
   ☒ **A, B,** and **D** are incorrect because they are true.
2. ☑ **B** is correct. The Enterprise JavaBeans (EJB) tier does host system-level services such as transaction management, concurrency control, and security.
   ☒ **A, C,** and **D** are incorrect because they are true.
3. ☑ **D** is correct. When using a compensating transaction, it is not always possible to undo the effect of a committed transaction, even if the server crashes.
   ☒ **A, B,** and **C** are incorrect because they are true.

Explain the Use of Patterns in the JEE Framework

4. ☑ **C** is correct. JEE uses the core patterns to enable flexible association of EJB classes with other components. The Designer pattern is not one of them.
   ☒ **A, B,** and **D** are incorrect because they are true. Decorator, factory, and proxy are core patterns to enable flexible association of EJB classes with other components.

Describe the Concepts of "Best Practices" and "Guidelines

5. ☑ **C** is correct. A session bean facade *does* shield the client from complex entity bean relationships and manages workflow on the client's behalf.
   ☒ **A, B,** and **D** are incorrect because they are true.

Review Best Practices Applicable for All Tiers

6. ☑ **A** and **D** are correct. Enterprise JavaBeans are best used with large and complex enterprise applications with high deployment and transactional requirements.
   ☒ **B** and **C** are incorrect.
7. ☑ **D** is correct**.** The security tier is not considered a JEE tier.
   ☒ **A, B,** and **C** are incorrect because they are true—client, web, and EIS are JEE tiers.
8. ☑ **B** is correct. Coding data manipulation logic, format and display code, and user event handling together can complicate and make application maintenance problematic and costly.
   ☒ **A, C,** and **D** are incorrect because they are true.

9. ☑ **D** is correct. Does not necessarily simplify database design and typically, the model and not the view component accesses the database.
☒ **A, B,** and **C** are incorrect because they are true.

10. ☑ **C** is correct. The View and not the Model determines how the results should be displayed.
☒ **A, B,** and **D** are incorrect because they are true.

Explain How to Apply MVC to the Web Tier

11. ☑ **B** is correct. The Calculator is not a component of the MVC pattern.
☒ **A, C,** and **D** are incorrect because they are true, as Model, View, and Controller are the components of the MVC pattern.

12. ☑ **D** is correct. The Model, not the Controller, maintains the application state.
☒ **A, B,** and **C** are incorrect because they are true.

Review the Best Practices for the Presentation Layer

13. ☑ **D** is correct. A misappropriation that does not compromise availability is not typically considered a threat to enterprise-critical assets.
☒ **A, B,** and **C** are incorrect because they are true. The disclosure of confidential information, the modification or destruction of information, and the misappropriation of protected resources are typically considered threats to enterprise-critical assets.

Illustrate When to Use JEE Technology for Given Situations

14. ☑ **C** and **D** are correct. When using screen scrapers, any changes to the legacy user interface will also affect the new GUI. In addition, screen scraping is the best alternative only if the existing UI is tightly coupled with the business tier of the legacy application. Therefore, choices **C** and **D** are false and, therefore, the correct choices.
☒ **A** and **B** are true about screen scrapers and, therefore, the incorrect choices.

15. ☑ **D** is correct. Session beans can be used for making both the CORBA call for block purchase of telephone numbers and the custom synchronous call to request a special vanity number.
☒ **A, B, C, E,** and **F** are incorrect. Both operations represent business processes involving partner OSS integration. Applets are not used for modeling the business workflow of a system. Therefore, choice **A** is incorrect. JSP represents the view construction process in an MVC application. It should not be used for processing business logic. Therefore, **B** is incorrect. Entity beans represent the business model of an application and provide a representation of enterprise data. They are not to be used for workflow processing, which is better accomplished by using

session beans. Therefore, **C** is incorrect. JNDI provides Naming and Directory interfaces, not workflow processing. Therefore, choice **E** is incorrect. The question specifically says that a synchronous mechanism is to be used for the vanity number request. The CORBA RPC call for TN reservation is also synchronous. MQ Series is a MOM used for messaging. Messaging is an inherently asynchronous communication mechanism. Therefore, choice **F** is incorrect.

16. ☑ **A** is correct as using JSP and servlets is the best option.
☒ **B, C, D,** and **E** are incorrect. The important element to this question is that the revenue is generated by click-through sales. This implies that there are no transactions involved and you do not need to use EJBs. Therefore, choices **B** and **C** are not the best options. Perl/CGI scripts are harder to maintain than Java code. Therefore, choice **E** is not the best option.

17. ☑ **B** is correct. The JEE Connector architecture provides a Java solution to the problem of connectivity between the many application servers and most EISs, not just those already in existence.
☒ **A, C,** and **D** are incorrect because they are true.

18. ☑ **D** is correct. A Transaction Management contract *does* support transactions that are managed internal to an EIS resource manager without the necessity of involving an external transaction manager.
☒ **A, B,** and **C** are incorrect because they are true.

19. ☑ **D** is correct. Enterprise application development is about building an enterprise application from scratch or integrating new enterprise applications with applications and EISs that have been in operation for some time; they are often referred to as an enterprise's *legacy* systems.
☒ **A, B,** and **C** are incorrect because they are true.