| [Print](javascript:window.print()) | [Close](javascript:parent.window.close()) |

**Course Transcript**

Introduction to Java EE 6

**An Architectural Overview**

| [1. Introduction to Java EE 6](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#t3) |

| [2. Web Components](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#t55) |

| [3. Enterprise JavaBeans in Java EE 6](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#t56) |

| [4. Java Persistence API and Connectivity](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#t5) |

| [5. Understanding Java EE 6](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#t12) |

**Preparing a Basic Application**

| [1. The Java EE 6 Environment and Application Components](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#t15) |

| [2. Stepping Through a Basic Java EE 6 Application](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#t21) |

| [3. How to Create a Basic Java EE 6 Application](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#t24) |

Introduction to Java EE 6

Learning Objective

*After completing this topic, you should be able to*

* *describe the key features of Java EE 6*

**1. Overview of Java EE**

Business application requirements for enterprises are becoming increasingly complex in today's global marketplace. Companies work across continents and time zones over the Internet, doing business 24 hours a day, seven days a week. This often includes dealing with a variety of currencies, languages, and business practices.

The Enterprise Information Systems, or EIS, used by these companies need to be able to handle a large amount of in-house data, often stored in multiple locations. This data is processed in a variety of programming languages, each of which uses different routing protocols.

Problems can lead to a loss in profits. So enterprise systems and applications have to be robust, and adaptable enough to deal with new business situations. They need to be secure, scalable, and highly available.

Java Enterprise Edition, or Java EE, was originally designed to focus on distributed components. It has since been upgraded with a number of standard methods so it can handle new technical solutions, such as RESTful web services and SOAP.

The new specifications that comprise Java EE are aimed at enterprise applications. Java EE allows you to develop secure, transactional, distributed, and interoperable applications.

Java EE consists of a number of application programming interfaces, or APIs, that enable you to create standard component-based, multitier applications. Each component is deployed in a different container, with each container offering a series of services.

Java EE includes standard services or interfaces to handle

**messaging**

Messaging is handled using the Java Message Service, or JMS.

**persistence, and**

Persistence is handled using the Java Persistence API, or JPA.

**transactions**

Transactions are handled using the Java Transaction API, or JTA.

Java EE was created to support applications for implementing services in enterprises. The basis of the Java EE application model is the Java programming language and the Java Virtual Machine, or JVM.

Graphic

*A Java EE Application has three tiers: A client tier that consists of Dynamic HTML pages, a web and business tier that consists of JSP pages and Enterprise beans, and an EIS tier that consists of a database.*

Business functions are conducted in the middle tier to control and manage applications that support users such as employees, customers, and suppliers. This tier is an environment that's commonly housed on a dedicated server system that has access to all services in an enterprise.

The accessibility, scalability, and manageability that enterprise-level applications need is provided by the Java EE architecture model. This model defines the requirements for implementing services as multitier applications.

The processes that implement a multitier service are divided by the Java EE architecture model into three layers:

Graphic

*A Java EE Application has three tiers: A client tier, a web and business tier, and an EIS tier.*

* standard system services that the Java EE platform provides   
  *Standard system services include Naming, Security, Transaction management, and Message Service.*
* developer-implemented business and presentation logic, and   
  *The server-side presentation container on the web server consists of JSP, Java servlet, XML, Web service, and JEE platform. The Server-side business logic on the EJB container consists of EJB, Web Service, and JEE platform.*
* platform reliability, which provides solutions to major issues at the systems level during multitier service development   
  *A Deployment tier consists of EJB, JSP, HTML, and Java Servlet.*

Java EE 6 has a number of new features that make it

**lighter**

To make the platform lighter, the Java EE 6 expert group introduced profiles and Enterprise JavaBean Lite, or EJB Lite. Profiles make it possible to reduce the size of the platform to meet developers' needs. EJB Lite, a subset of the full EJB feature, focuses exclusively on interceptors, local interfaces, security, and transactions.  
  
Some features have also been proposed for removal when EE 7 is released. An example is Java EE Application Deployment, which received very limited vendor support.

**easier to use**

To inform a container about its behavior, a Java EE component requires metadata. This metadata previously took the form of XML in a deployment descriptor file, and code annotations. Java EE 6 includes new annotated classes such as renderers, servlets, JavaServer Faces converters, validators, and JavaServer Faces managed beans. Each class includes optional XML deployment descriptors. This makes the web tier easier to use.

**richer, and**

Java EE 6 is richer and more versatile than previous versions. It adds the new Java API for RESTful Web Services, or JAX-RS, specification, along with an improved version of the persistence API. The development and packaging of EJBs is easier, and they include new features, such as a timer service for task scheduling and an asynchronous call feature.  
  
A new singleton session has also been added. You now need only one annotation to turn a Java class into a container-managed singleton.

**more portable**

Java EE 6 is more portable, with a common syntax used across application servers for Java Naming and Directory Interface, or JNDI, names. Code and configuration files no longer have to be changed for an application's deployment to different application servers. An EJB now includes an embedded container so it can be used in the Java Standard Edition, or Java SE environment. The container is a standard API that enables the EJB to execute in Java SE.

To be Java EE 6 compliant, an application server has to comply with particular specifications.

Supplement

*Selecting the link title opens the resource in a new browser window.*

**Job Aid**

Access the job aid [Java EE 6 Specifications](javascript:doWindow('./jajteo_a01_it_enus_t301_frame.html')) for more information about the specifications.

Question

Identify the features of Java EE 6.

**Options:**

1. Profiles have been added to make the platform smaller
2. A new singleton session is included
3. An EJB includes an embedded deployment descriptor file for use in Java SE environments
4. Validation services are improved with EJB Lite

Answer

***Option 1:****Correct. In addition to EJB Lite, profiles have been introduced in Java EE 6 to enable the size of the platform to be reduced to more efficiently meet the needs of developers.*

***Option 2:****Correct. A new singleton session has been added, so you need only one annotation to turn a Java class into a container-managed singleton.*

***Option 3:****Incorrect. EJBs now include embedded containers, rather than deployment descriptor files, so they can be used in a Java SE environment.*

***Option 4:****Incorrect. EJB Lite, a subset of the full EJB feature, focuses exclusively on interceptors, local interfaces, security, and transactions. It doesn't improve validation services.*

**Correct answer(s):**

1. Profiles have been added to make the platform smaller  
2. A new singleton session is included

**2. Java EE architecture**

Java is not just a high-level, object-oriented programming language – it's also an environment platform used for running Java applications.

A Java platform consists of a JVM, which is used to run Java technology applications. Each platform also includes a collection of software components in the form of an API. The API is used to create applications or other software components.

Together the JVM and API enable applications written for a specific Java platform to run on any compatible system.

The core functionality of the Java programming language is provided by Java SE, which defines basic language elements, such as objects and types.

The Java SE platform includes development tools and deployment technologies along with the core API and JVM. With a number of toolkits and class libraries, it can be readily used for creating Java applications.

The Java EE platform is built on top of the Java SE platform. It has an API and additional libraries that enable you to develop and deploy large-scale, fault-tolerant, multitier network applications that are secure and scalable.

Java EE uses open standards that are implemented by commercial frameworks such as Websphere, as well as open-source frameworks such as GlassFish.

These standards are used to handle stateful components, object persistence, and security, for example. You can deploy an application with minimal changes to application servers that conform to these standards.

Specifications in the Java EE platform are made up of numerous Java Specification Requests, or JSRs. JSRs are actual descriptions of proposed or finalized specifications intended for use with the platform.

Java EE architecture specifications are implemented using logical domains called *containers*. A container is a Java EE runtime environment.

Containers function as a link between components and the underlying platform-specific services that support them – for example, by supporting API sets or offering database access services to components. Because Java EE is built on the Java SE platform, Java EE components are able to use Java SE APIs.

Well-defined contracts enable components to communicate with both the Java EE infrastructure and each other. Components need to be packaged in an archive before deployment in a container.

There are four types of containers:

**applet**

Applet containers execute applet components, and are contained in most web browsers. This type of container provides a secure environment during applet development because it uses a sandbox security model. Code executed within the sandbox is restricted by the container from accessing local resources, such as system files or processes.

**web**

Underlying services for executing and managing web components are provided by the web container. This includes EJB Lite, JavaServer Faces pages, and web services. Web pages are fed to client browsers by the web container. It instantiates, initializes, and calls servlets, and supports the HTTP and HTTPS protocols.

**EJB, and**

New EJB instances are created by the EJB container. It executes and manages enterprise beans, which contain the business logic tier of Java EE applications. It also manages enterprise bean life cycles, and provides distribution, transaction, naming, and security services, for example.

**application client**

The application client container, or ACC, contains files such as a set of Java classes and libraries. You need these files to integrate naming services, security management, and injection into Java SE applications.

An implementation has to support four component types defined by the Java EE runtime environment:

**applets**

Applets are GUI applications that use the feature-rich Swing API. This API is used to create powerful user interfaces that can be executed in web browsers.

**applications**

Applications are client-executable programs such as batch-processing programs or GUIs. They can access the facilities of the Java EE middle tier.

**enterprise applications, and**

Enterprise applications are made up of elements such as EJBs, the Java Transaction API, asynchronous calls, and JMS. Enterprise applications are executed in EJB containers. EJB components process transactional business logic and are managed by containers.

**web applications**

Web applications manage HTTP requests received from web clients. They are executed in web containers and consist of servlets, servlet filters, and JSP pages, for example. RESTful web service and SOAP endpoints are supported by servlets. Web applications can contain EJB Lite since Java EE 6.

The components you use depend on the type of application you are creating, and can be combined to increase functionality.

For example, if you want to build a web application that remotely calls the business tier and also uses messaging, you can combine the EJB and web containers.

Graphic

*A web container consisting of JSP page and Servlet is combined with an EJB container consisting of Enterprise beans.*

The Java EE platform environment provides a number of services for supporting deployed components. Applications on the application server are responsible for ensuring that service requirements for these components are met.

Java EE services include

**Java Transaction API, or JTA**

The JTA provides a transaction demarcation API, and an interface between the transaction manager and a Service Provider Interface level resource manager. It's used by both the application and the container.  
*A Transaction manager is connected to a number of Resource managers, one of which is connected to an application.*

**Java Persistence API, or JPA**

The standard API for object-relational mapping is JPA. You can query objects located in the underlying database using the JPA's Java Persistence Query Language.  
*A client connects to an EJB container consisting of session and entity beans. It also connects to a database via the JPA and persistence.xml.*

**validation**

Validation provides a constraint declaration and validation service at the Bean class level.  
*A client connects to a database via a Java Validation container consisting of presentation layer, business layer, and data access layer tiers.*

**JMS**

The JMS enables asynchronous communication between components via messages. Both point-to-point and publish-subscribe messaging models are supported.  
*A connection factory, which connects to a session container is instantiated. The session connects to a message, a message producer, and a message consumer. The producer and consumer are in turn connected to Destination databases.*

**Java Naming and Directory Interface, or JNDI**

Naming and directory systems are accessed using the JNDI API. Applications use it to bind names to objects, making it easier to find the objects in a directory. This service is included in Java SE.  
*A java application is connected to the JNDI API. A naming manager interfaces between the JNDI API and a JNDI SPI, which consists of LDAP, DNS, and New.*

**JavaMail, and**

The JavaMail API enables applications to send e-mails.  
*A Java application consisting of a filter and SMTPHandler connects to a mail client via a database server.*

**JavaBeans Activation Framework, or JAF**

A framework for managing data in various MIME types is provided by the JavaBeans Activation Framework, or JAF, API. It's used by JavaMail and is also included in Java SE.  
*A Java application connects to SMTP server, IMAP mail store, POP3 mail store via JAF.*

Other Java EE services include

**Java API for XML Processing, or JAXP**

**Java EE Connector Architecture, or JCA**

EIS components, such as a mainframe, can be accessed from a Java EE component using the Java EE Connector Architecture, or JCA.  
*An IES server connects to a Java EE component via JCA.*

**Java Authentication and Authorization Service, or JAAS**

The Java Authentication and Authorization Service, or JAAS, provides security services to other services. These services are then able to enforce user authentication and access controls.  
*A JAAS component is connected to a LoginModule container for user authentication and access control.*

**web services**

The SOAP/HTTP protocol is supported by the Java API for XML Web Services, or JAX-WS. It replaces the Java API for XML-based RPC, or JAX-RPC. REST-style web services are supported by the Java API for RESTful Web Services, or JAX-RS.  
*A client JAX-WS runtime connects to a service runtime using a SOAP message.*

**Java Management Extensions, or JMX API**

Server and container management APIs are defined using a specialized management enterprise bean in Java EE. Some management support is also supplied by the Java Management Extensions, or JMX API.  
*Enterprise bean management consists of server management and container management.*

**Dependency Injection for Java, or DI, and**

Resources, such as persistence units and datasources, have been able to be injected into managed components since Java EE 5. The Dependency Injection for Java, or DI, and the Context and Dependency Injection, or CDI, specifications build on this service in Java EE 6.  
*EIS, Persistence Units, and Datasources are injected into managed components.*

**Java EE Deployment Specification**

Deployment contracts are defined by the Java EE Deployment Specification. This standardizes application deployment between Java EE products and deployment tools.  
*An assembly root consists of Web module and a number of META-INF containers, one of which is connected to an application.*

Java EE supports Remote Method Invocation, or RMI, protocols to makes calls to deployed components held in containers. For example, EJBs can be accessed either locally or remotely using RMI.

Java Remote Method Protocol, or JRMP, is the native RMI protocol for Java SE. You can use RMI to call remote objects no matter what the underlying protocols are.

The Remote Method Invocation over Internet Inter-Orb protocol, or RMI-IIOP, is an extension of RMI. It is used to integrate with the Common Object Request Broker Architecture, or CORBA. ACC uses RMI-IIOP to communicate with the EJB container, for example.

As long as external CORBA objects include the IIOP protocol, Java EE application components can call them using the Java interface description language, or IDL. These objects can be written in a number of languages, including C++ and Java.

Java EE also supports the use of HTTP protocols to deploy container components. ACC uses HTTP to communicate with the web container, for example.

The java.net package in Java SE defines the client-side API. Servlets, JSF interfaces, JSPs, and SOAP and RESTful web services define the HTTP server-side API.

Java EE supports HTTPS, which is a combination of HTTP with the Secure Sockets Layer, or SSL, protocol.

Question

Identify characteristics of Java EE components.

**Options:**

1. Components in containers are accessed using Remote Method Invocation
2. Components use containers to link to platform-specific services
3. Service requirements for deployed components are   
   provided by servlets
4. Web application components manage RMI-IIOP requests from web clients

Answer

***Option 1:****Correct. Java EE supports the use of RMI protocols to access components locally or remotely.*

***Option 2:****Correct. Containers function as a link between components and the underlying platform-specific services that support them.*

***Option 3:****Incorrect. Applications on the application server are responsible for ensuring service requirements for deployed components are met.*

***Option 4:****Incorrect. Web application components manage HTTP requests, rather than RMI-IIOP requests, from web clients.*

**Correct answer(s):**

1. Components in containers are accessed using Remote Method Invocation  
2. Components use containers to link to platform-specific services

**3. Summary**

Java EE includes standardized methods to deal with the requirements of creating enterprise applications. It's become lighter thanks to the introduction of platforms and EJB Lite. It's also easier to use with new annotated classes such as renderers, and servlets. Java EE 6 is richer and more versatile. Additions include simpler packaging for EJBs, a new singleton class, and the addition of the JAX-RS specification. Portability has increased with a common syntax used across application servers for JNDI names. Embedded containers in EJBs have been added for use in the Java SE environment.  
  
The Java EE platform is built on top of the Java SE platform. Java EE architecture specifications are implemented using logical domains called containers, which provide services to hosted components. These services include validation, security, and XML processing. Java EE uses RMI protocols to access components held in containers. It also supports the HTTP and HTTPS protocols.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#top)

Web Components

Learning Objectives

*After completing this topic, you should be able to*

* *match the client side Java application components to how they can be used*
* *recognize when different types of web services should be used*

**1. Java client-side components**

**Drill Down Home Page**

In general, web components are designed to facilitate communication between client and server applications over the Web. There are a number of Java components you can use to support client applications.

**Page 1 of 5: Java Servlets**

Java Servlet technology is a platform-independent way of providing dynamic content to users, and runs on the server side.

Graphic

*A web service container consisting of JSP and a servlet connect to a web service, a database, and a web page. The web page in turn is connected to a client web browser.*

It addresses some of the shortcomings of the Common Gateway Interface, or CGI, scripts, which are platform-dependent and lack scalability to support a growing number of requests.

**Page 2 of 5: Java Servlets**

A servlet is a class used in Java programming that responds to requests to extend applications hosted on a server. They are generally used for web services and use HTTP-specific classes, but can be incorporated into any request-response environment.

Java Servlets use the javax.servlet and javax.servlet.http packages and implement the built-in interfaces and classes.

**Page 3 of 5: Java Servlets**

Servlets also

* implement the servlet interface
* extend GenericServlet for generic services, and
* handle HTTP-specific services using the doPost and doGet methods of the HttpServlet class

**Page 4 of 5: Java Servlets**

When a request is mapped to a servlet, the life cycle starts with checking that the servlet has been instantiated. If it hasn't, the servlet class is loaded and an instance of this class is created. The init method is then called to initialize the servlet.

Once a servlet instance is running, the service method is called using the request and response objects.

**Page 5 of 5: Java Servlets**

For example, you work for a tea import business that has a large database of cultivars and provides merchandise to wholesalers. You want to provide your clients with a way of filtering out all tea imported from a particular country.

To do this, you can create a servlet that dynamically retrieves content from the database and provides this information to users.

**Page 1 of 6: JavaServer Faces Technology**

You use JavaServer Faces, or JSF, to create server-side, Java-based web applications. JSF technology provides tag libraries that you use to add components to web pages, and connects server-side objects to these components.

It also includes an API used to manage components and their states. It can also validate and convert data, and handle events. Additionally, you can use JSF for internationalization and provide accessibility features for physically impaired users.

**Page 2 of 6: JavaServer Faces Technology**

JSF technology is very extensible, enabling you to build web pages and add components using component tags. You can also bind components to server-side data and customize components for repurposing.

JSF technology can also save and restore an application's state after a server request expires, and you can link server-side application code to component-generated events.

**Page 3 of 6: JavaServer Faces Technology**

A JSF application includes a number of components:

* a collection of web pages populated with components
* a collection of tags that enables new components to be added
* a collection of custom tags that enables custom objects to be added
* a collection of managed beans that supports a small selection of basic services, and
* a web.xml deployment descriptor

**Page 4 of 6: JavaServer Faces Technology**

Optionally, a JSF application can contain a set of custom objects such as listeners, validators, and converters. It can also include application configuration files to describe page navigation instructions and custom objects.

There are several benefits to using JSF technology. In JSF technology there's an architectural separation between web application processes and interfaces. This enables these parts to be developed separately by various developers, which is an advantage over traditional client-side applications.

**Page 5 of 6: JavaServer Faces Technology**

An HTTP request can also be mapped to a component-specific event handler, which enables objects on the server to remain stateful. This means that an object can be reserved for the user who originally requested it, and will be cleared only when the user no longer needs it.

JSF also enables you to use common component and web-tier concepts without being bound to a markup or scripting language.

**Page 6 of 6: JavaServer Faces Technology**

For example, you're hired as the senior technical lead at a large corporation. Your manager approaches you for a recommendation on which technology to use to create components that will extend their application.

In such a case, you suggest JavaServer Faces Technology because it enables you to extend existing applications.

**Page 1 of 3: Facelets**

Facelets is a page declaration language you use to create JavaServer Faces views and component trees. You use HTML-type templates for this purpose.

Facelets includes features for using XHTML to create web pages, and supports the Expression Language, or EL. In addition to supporting the JSF and JSTL tag libraries, Facelets has its own tag library and features templates for use with pages and components.

**Page 2 of 3: Facelets**

The main advantage of using Facelets is the time you save during the development and deployment processes. One reason for this is the fact that code can be reused through templates and creating composite components.

Facelets also enables you to customize server-side components and objects by extending them to provide more features to users. Facelets is quick to compile and the EL is validated while being compiled. Rendering Facelets is also a quick process.

**Page 3 of 3: Facelets**

Consider extending an existing application using JavaServer Faces. In order to accelerate the roll-out to users, you can use Facelets to provide HTML templates to provide a front-end for users.

**Page 1 of 4: Java Server Pages**

You use Java Server Pages, or JSPs, to create web pages that are dynamically generated based on client requests. This occurs on the server that compiles them as servlets. JSPs are similar to HTML and XHTML pages with the additional use of tags that trigger server-side processes that invoke Java code on the server.

Graphic

*A web service container consisting of JSP and a servlet connect to a web service, a database, and a web page. The web page in turn is connected to a client web browser.*

The servlet API plays a central role in JSP on which many of the processes are based. Through servlets, the server can accept client-based HTTP requests and return dynamic responses. Like servlets, JSPs make use of server-side resources such as databases and web services.

**Page 2 of 4: Java Server Pages**

JSPs run in a servlet container on the server and direct the JSP life cycle in various ways:

* by compiling the JSP code into a servlet
* by loading and initializing the servlet
* by reading client requests and directing them to the JSP servlet
* by returning the HTML and XHTML tags to the client for browser display, and
* by unloading the JSP servlet and blocking any further requests

**Page 3 of 4: Java Server Pages**

XHTML follows the XML rules, which you can enforce by using the .jspx file extension. You can use only HTML tags to create the JSP, in which case you use the .jsp file extension.

Using XHTML, however, enables you to add XML-type tags to create more dynamic content. According to the JSP specification you can use three types of elements – Directive, Scripting, and Action Elements.

**Page 4 of 4: Java Server Pages**

For example, you work for an insurance company that provides benefits enrollment to their customers online. You want to provide them with a summary of their benefits after enrollment, which contains details of their coverage and the total monthly out-of-pocket expense they will incur.

To accomplish this you can use Java Server Pages to dynamically generate the users' pages, in conjunction with Java Servlets to retrieve the dynamic content from the web server.

Question

Match the Java server-side components to the scenarios in which their use is appropriate.  
  
More than one component may match to each scenario and the components may be used in conjunction to perform the required tasks.

**Options:**

1. Java Servlets
2. JavaServer Faces
3. Facelets
4. Java Server Pages

**Targets:**

1. You work for a large clothing retailer and want to provide a way for your customers to make online purchases and browse through a wide variety of items you carry in store. You want to use Java technologies that enable users to retrieve dynamic data from the server and display it in a dynamically generated web page.
2. You work for a large corporation with hundreds of employees worldwide. Your manager has asked you to incorporate an employee listings page on the corporation's intranet web site. This page should display basic contact information for each employee. You want to extend the existing intranet web site for this purpose.

Answer

*You can use Java Servlets to retrieve dynamic data from a server, and you can use Java Server Pages to dynamically generate the web page that displays the data.*

*You can use JavaServer Faces to extend the existing intranet web components, and Facelets to generate the front-end using HTML templates.*

**Correct answer(s):**

Target 1 = Option A, Option D

Target 2 = Option B, Option C

**2. Using web services**

Web services are extensible client and server applications that provide standard communication between software applications, using HTTP as the communication medium. Web services operate irrespective of the platform or framework used on either end.

Web services use XML to enable a client or server to read and interpret the description of the service. In the context of web services, a client is also referred to as a service consumer, and a server is also referred to as a service provider.

There are two main types of web service you can implement – JAX-WS and RESTful web services.

JAX-WS can be implemented by using JAX-WS technology, which uses the Simple Object Access Protocol, or SOAP, standard to send information using XML. This type of web service uses the Web Services Description Language, or WSDL, to describe operations offered by the service. WSDL is an XML-based language that can describe an interface.

At minimum, a SOAP-based service must include three elements. The first is a contract written in WSDL that describes the interface offered by the web service. This can include descriptions of operations, the location of the web service, bindings, and relevant messages.

Second, the web service architecture must be able to comply with non-functional requirements such as transaction, security, trust, and addressing. And third, it should support asynchronous processing and invocation. This support can be provided through the use of the JAX-WS API or the Web Services Reliable Messaging, or WSRM, standard.

Representational State Transfer – or RESTful – web services can be implemented using the JAX-RS technology. They don't require the WSDL service API definitions that JAX-WS do, and are better suited to basic web service needs.

JAX-RS can be implemented using a specification called Project Jersey, which supports the JAX-RS annotation for building web services with Java or the Java Virtual Machine – or JVM. RESTful web services use a simple infrastructure and make use of existing W3C and Internet Engineering Task Force, or IETF, standards.   
  
This makes for a quick and inexpensive development cycles. You can use the NetBeans IDE, for example, to quickly develop RESTful web services with reduced complexity.

JAX-WS are best used when you're integrating enterprise-level applications that have to conform to high Quality of Service, or QoS, requirements.  
  
JAX-WS is best in this regard because it has built-in support for the WS set of standards and protocols, which ensure reliable and secure communication between servers and clients.

You've been hired by a large banking firm and are working on a complex application that processes online banking transactions. The application supports many services, such as transfers between accounts, payments on credit card balances, and bill payment services.

This application is extremely complex so JAX-WS, with XML messages that follow the SOAP standard, would be the best technology to use.

RESTful web services are best used for simple web service integration over the web scenarios. Using JAX-RS, you can create scalable services that can grow as demand increases. You can also update a server without fear of introducing incompatibilities from the client side of the connection.  
  
JAX-RS includes built-in components such as HTTP routers that help you keep the architecture simple and make it unnecessary to develop complex procedures.

RESTful web services are generally open to a wide range of clients that need to consume only certain components of the services. They also enable clients to mix and match components from different services.

You are a small business owner and would like to create a blog site to drive customer interest to your web site and create more revenue. Blog sites are considered RESTful because they involve downloading XML files in the Really Simple Syndication, or RSS format, or the Atom format, which contain lists of links to other resources.

Question

Match each web service type to the scenario in which you would use it.

**Options:**

1. RESTful web services
2. JAX-WS

**Targets:**

1. You're working for a small but expanding company that requires their employees to interact on a daily basis. They decide to integrate social networking functionality into their existing web infrastructure. You plan to use the NetBeans IDE for this purpose, and hope for a short development life cycle.
2. You've been hired as a programmer for an international travel agency offering flights and vacation packages online. The online application will be complex and requires coordination of multiple external parties. It should also support a hold feature where a customer can hold one or more airline seats for 5 minutes before deciding whether or not to purchase.

Answer

*You can use RESTful web services to add this functionality – the scalability of REST will facilitate a growing number of users.*

*You can use JAX-WS for this application because this type of web service is best suited for enterprise application integration.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

**3. Summary**

You can use a number of Java application components to provide services to clients. You use Java Servlets to provide dynamic content to users irrespective of the platform they are using. You use JavaServer Faces to create server-side web applications. You use facelets to provide users with views of JSF applications. And you use JSP to generate web pages dynamically based on client requests, using HTML or XHTML.  
  
There are two main types of web services you can use for Java Enterprise development. JAX-WS are most suitable when strict QoS rules must be enforced, and support for the WS set of standards and protocols is required. RESTful web services are best used when web services are integrated over the Internet. They provide built-in features and components that save time during development and deployment.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#top)

Enterprise JavaBeans in Java EE 6

Learning Objectives

*After completing this topic, you should be able to*

* *describe the benefits of using Enterprise JavaBeans*
* *distinguish between the different types of Enterprise JavaBeans*
* *match the type of enterprise bean to scenarios that fit their application*

**1. Benefits of Enterprise Java Beans**

An Enterprise JavaBean, or EJB, is a server-side component of the Java Enterprise Edition platform that runs in the EJB container. The EJB container provides system-level services for transactions and security, while the beans provide business logic to an application.

Graphic

*An EJB container consisting of enterprise beans is contained in the Java EE server layer. It interacts with a database layer and an application client container layer.*

This arrangement enables developers to concentrate on the business logic programming and lets the EJB container handle the transactions and security.

Business logic is the code that enables the business processes of an application. For example, an enterprise bean can implement methods used by a client application to book a hotel room.

Versions of EJB include

**EJB 1.0**

EJB version 1.0 supported stateful and stateless session beans. Home and remote interfaces were used.

**EJB 1.1**

EJB version 1.1 added support for entity beans. It used the XML deployment descriptor. It also introduced roles to support application assembly and deployment.

**EJB 2.0**

EJB version 2.0 used local interfaces to reduce the necessity of passing arguments by value. It also supported relationships and a query language with entity beans.

**EJB 2.1**

EJB version 2.1 supported web services with the Simple Object Access Protocol, or SOAP and HTTP.

**EJB 3.0, and**

EJB version 3.0 focused on ease of use. It replaced the entity bean specification with Java Persistence API, or JPA, and introduced dependency injection, interceptors, and life cycle callbacks.

**EJB 3.1**

EJB version 3.1 was released in 2009 with Java Enterprise Edition, or Java EE, 6.

There are three benefits of using EJBs:

* system-level services are provided by the EJB container
* the bean contains the business logic and as a result, creates thinner clients, and
* new applications can be built from existing beans

Whether you should use EJBs depends on the requirements of your application. Generally you should consider using EJBs if your application must be scalable, if transactions must ensure data integrity, or if your application must support a variety of clients.

Question

What are some of the benefits of using EJBs?

**Options:**

1. System-level services are handled by the EJB container
2. The clients contain the business logic
3. New applications can be built from existing beans
4. Relationships and a query language are supported by entity beans

Answer

***Option 1:****Correct. The EJB container handles transactions and security, enabling you to focus on business logic.*

***Option 2:****Incorrect. A benefit of using EJBs is that the beans contain the business logic, and as a result, produce thinner clients.*

***Option 3:****Correct. You can take a modular approach to building new applications using existing beans.*

***Option 4:****Incorrect. Entity beans were replaced with JPA in EJB version 3.0.*

**Correct answer(s):**

1. System-level services are handled by the EJB container  
3. New applications can be built from existing beans

**2. Types of Enterprise Java Beans**

There are two types of enterprise beans – session beans and message-driven beans.

Graphic

*A middle tier EJB container consists of Session beans and Message-driven beans. This container connects to the web container in the same tier, the client application in the client tier, and a database in the database tier.*

**Drill Down Home Page**

A session bean performs a user task, whereas a message-driven bean acts as a listener.

**Page 1 of 5: Session beans**

Methods in a session bean are invoked by a client and are not persistent. A bean performs server-side processing when it's invoked over local, remote, or web-service client views.

**Page 2 of 5: Session beans**

A stateful session bean consists of instance variables that represent the state of the client-bean session. This is also known as the conversational state.

A stateful session bean can connect only to one client at a time and it terminates the state when the session is finished.

Stateful session beans can't implement a web service. For this purpose, you can use a stateless session bean.

**Page 3 of 5: Session beans**

A stateless session bean contains a state-specific instance variable for as long as its methods are invoked. The client state is reset when the called method is executed.

Pooled stateless beans can have their instance variable state changed until the next time their methods are invoked.

The EJB container assigns stateless bean instances to one of multiple clients. The state can apply to all clients because all instances of the stateless bean are equivalent.

**Page 4 of 5: Session beans**

Stateless session beans enable scalability in applications that serve large numbers of clients.

Stateless session beans can handle more clients than stateful session beans and can implement a web service.

**Page 5 of 5: Session beans**

If you have an application with a bean that's accessed by multiple clients simultaneously, you use a singleton session bean.

There is one singleton session bean per application. It's instantiated when the application is started so it can perform initialization tasks. It can also perform clean-up tasks when the application shuts down.

Singleton session beans maintain their state between client sessions, but a new state will be invoked if the server shuts down.

**Page 1 of 6: Message-driven beans**

A message-driven bean acts as an event listener for Java Message Service, or JMS, event messages, or other messages sent from a Java EE component or a JMS application. Java EE applications use message-driven beans to perform asynchronous message processing.

**Page 2 of 6: Message-driven beans**

A message-driven bean has a number of characteristics:

* it executes when it receives a client message
* it's invoked asynchronously
* it has a short life cycle
* it can access shared data in the database
* it's stateless, and
* it's transaction aware

**Page 3 of 6: Message-driven beans**

Unlike a session bean, a message-driven bean doesn't have an interface through which clients can access it. A message-driven bean consists only of a bean class.

**Page 4 of 6: Message-driven beans**

A message-driven bean is similar to a stateless session bean in that

* no client-session state is maintained
* the EJB container can assign message-driven bean instances to one of multiple messages because all instances of the message-driven bean are equivalent, and
* messages from multiple clients can be processed by a single message-driven bean

**Page 5 of 6: Message-driven beans**

Message-driven beans consists of instance variables that can represent the state for handling messages.

**Page 6 of 6: Message-driven beans**

A message-driven bean is invoked when a client sends a message to the object for which the bean acts as a listener. The message can be delivered in a transaction context to improve reliability.

The JMS container calls the message-driven bean's onMessage method to handle the message.

The message is cast into one of the five message types and the business logic is applied to it. The bean can invoke helper methods and it can invoke a session bean.

Question

Match each feature to the corresponding bean type. More than one feature may match to each type.

**Options:**

1. Instance variables represent the state of the client-bean session
2. The bean connects to one client at a time
3. The bean terminates the state when the session is finished
4. The bean acts as an event listener
5. The bean executes when it receives a client message
6. The bean is invoked asynchronously

**Targets:**

1. Session bean
2. Message-driven bean

Answer

*A session bean maintains state, connects to a single client, and terminates when the session ends.*

*A message-driven bean acts as an event listener, executes when a message is received, and is invoked asynchronously.*

**Correct answer(s):**

Target 1 = Option A, Option B, Option C

Target 2 = Option D, Option E, Option F

**3. Enterprise bean applications**

**Drill Down Home Page**

The type of enterprise bean you use depends on a number of factors.

**Page 1 of 3: Stateful session bean**

You're developing an application that uses an enterprise session bean to interact with an accounting system. The bean needs to be able to handle additions to and subtractions from a client's account balance.

**Page 2 of 3: Stateful session bean**

There are a number of reasons to use a stateful session bean in this case:

* the bean's state must represent a bean-client session
* the bean's state must be maintained for all its method invocations
* the bean must act as an interface for a client and application components, and
* the bean must manage a number of enterprise beans' workflows

**Page 3 of 3: Stateful session bean**

To start creating the session bean class, you annotate the source code as @Stateful.

Code

package ejbSample.stateful;  
  
import javax.ejb.Stateful;  
import javax.ejb.Remote;    
import javax.ejb.Remove;   
import javax.ejb.\*;  
  
@Stateful(name="AccountBean")  
@Remote(AccountRemote.class)    
    public class AccountBean implements AccountRemote {float total = 0;

Then you implement the methods required to perform the business logic – in this case, the add and subtract methods. You end the state invoked by a client using a remove method.

Code

    public float add(float amount){  
    total += amount;  
    return total;  
    }  
    public float subtract(float amount){  
    total -= amount;  
     return total;  
    }  
@Remove    
    public void remove() {  
    total = 0;  
   }  
}

**Page 1 of 2: Stateless session bean**

Consider creating a session bean that can perform basic mathematical calculations for a client. To start, you create the business interface for the bean to define the methods that a client can call.

You annotate the source code for the bean class as @stateless.

Code

package net.sample.ejb3.stateless;  
  
import java.math.\*;  
import javax.ejb.Stateless;  
import javax.ejb.Remote;    
  
@Stateless(name="CalculatorBean")  
@Remote(CalculatorRemote.class)

And you implement the business methods defined in the business interface.

Code

public class CalculatorBean implements CalculatorRemote{  
  public float addition(float a, float b){   
    return a + b;  
  }  
  public float subtraction(float a, float b){  
    return b - a;  
  }  
  public float multiplication(float a, float b){  
    return a \* b;  
  }  
  public float division(float a, float b){  
    return b / a;  
 }  
}

**Page 2 of 2: Stateless session bean**

You should use a stateless session bean in this scenario for three reasons:

* the bean state doesn't contain client-specific data
* a single invocation must cause the bean to perform a generic task for all clients, and
* the bean must implement a web method

**Page 1 of 2: Singleton session bean**

Consider creating a session bean that counts how many hits a web page has received.

You annotate the source code for the bean class as @Singleton. The bean consists of a single method that counts the number of times a web page is accessed.

Code

package counter.ejb;  
  
import javax.ejb.Singleton;  
  
@Singleton

You can create a web front-end for the bean by creating a JavaServer Faces managed bean.

Code

public class CounterBean {  
    private int pageHits = 1;  
  
    public int getPageHits() {  
        return pageHits++;  
    }  
}

**Page 2 of 2: Singleton session bean**

There are a number of reasons to use a singleton session bean in this scenario:

* the bean state is common for the entire application
* multiple threads must be able to access the bean concurrently
* the bean needs to be instantiated when the application starts up, and
* the bean needs to be able to implement a web service

**Page 1 of 2: Message-driven bean**

Consider creating a bean that extracts text from incoming asynchronous messages and parses it to an application.

To start, you annotate the bean class code as @MessageDriven specifying the message queue to monitor.

Code

package mdb;    
   import javax.ejb.\*;  
   import javax.ejb.MessageDriven;  
   import javax.jms.Message;  
   import javax.jms.MessageListener;  
   import javax.jms.ObjectMessage;  
   import java.text.\*;  
   import javax.naming.\*;  
   import java.util.logging.Logger;  
  
 @MessageDriven(mappedName="jms/Queue")

You define the class as public and include a public constructor with no arguments. Optionally, you can include the MessageListener interface, which contains the method that performs the business logic required for the message.

Code

public class SimpleMessageBean implements MessageListener {  
    @Resource  
    private MessageDrivenContext mdc;  
    public void onMessage(Message theMessage) {  
      TextMessage txtMsg = null;  
      try {  
        if (theMessage instanceof TextMessage) {  
            txtMsg = (TextMessage) theMessage;  
            logger.info("The message is: " +  
                txtMsg.getText());  
        } else {  
            logger.warning("The message is the wrong type: " +  
                theMessage.getClass().getName());}  
      } catch (JMSException e) {  
        e.printStackTrace();  
        mdc.setRollbackOnly();  
      } catch (Throwable te) {  
        te.printStackTrace();  
      }  
   }  
}

**Page 2 of 2: Message-driven bean**

There are two reasons to use a message-driven bean in this case:

* the bean needs to be executed when a message arrives, and
* messages need to be received asynchronously

Question

Match each scenario to the type of enterprise bean that should be used.

**Options:**

1. The bean needs to add or remove items from a shopping cart
2. The bean needs to print a simple message to the user
3. The bean must display a daily price list
4. The bean must trigger a configuration update by sending a message

**Targets:**

1. Stateful session bean
2. Stateless session bean
3. Singleton session bean
4. Message-driven bean

Answer

*You use a stateful session bean in this case to maintain state between client requests.*

*You use a stateless session bean in this case because a retained state is not required.*

*You use a singleton session bean in this case to automate a process.*

*You use a message-driven bean in this case because you want to use a message to trigger an event.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

**4. Summary**

An EJB is a server-side component of the Java EE platform. The EJB container provides system-level services for transactions and security, while the beans provide business logic to an application. There are three benefits of using EJBs – system-level services are provided by the EJB container, the bean contains the business logic resulting in thinner clients, and new applications can be built from existing beans.  
  
There are two types of enterprise beans – session beans and message-driven beans. Methods in a session bean are invoked by a client and aren't persistent. Java EE applications use message-driven beans to perform asynchronous message processing.   
  
You can use a stateful or stateless session bean, a singleton session bean, or a message-driven bean depending on what kind of application you're developing and how the bean will be used.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#top)

Java Persistence API and Connectivity

Learning Objectives

*After completing this topic, you should be able to*

* *describe the key characteristics of JPA 2.0*
* *identify how to use JPA in a given scenario*

**1. Java Persistence API 2.0**

Java Persistence technology is a method of permanently storing application data. In the past, data could be stored using relational databases, serialization, or Java Database Connectivity – or JDBC.

These methods had shortcomings that were addressed through object-relational mapping tools, or ORM, which are implemented in Java EE 6 through the Java Persistence API, or JPA.

ORM works on the principle of external applications retrieving data from a relational database by mapping objects to the relational data. This data is then displayed in an object-oriented view, which also enables the data to be modified and written back to the database.

Java Persistence technology consists of four main features:

* the JPA
* the Java Persistence Criteria API
* a query language, and
* object-relational mapping metadata

The JPA enables you to perform the standard create, read, update, and delete operations on a relational database. You can retrieve information from the database using the Java Persistence Query Language, or JPQL.

The Java Transaction API, or JTA, enables you to lock data retrieved from a database at an external location. This ensures that while a user is busy with a set of relational data, another user can't retrieve and change that data, thereby preventing data anomalies. JPA also supports locking data on local resources.

The concept of ORM was first used in a product called Smalltalk, before Java was invented. ORM was first standardized in Java through Java Data Objects, or JDOs.

Persistence was later also addressed through the use of Entity Container Managed Persistence or CMP, and incorporated in Enterprise Java Beans, or EJB 1.0. Although this was effective to some extent, it provided limited persistence functionality.

JPA 1.0 was introduced in Java Enterprise Edition, or Java EE, 5 and also bundled with EJB 3.0. JPA 2.0 improved on this in that it facilitates development and includes new features. In Java EE 6, JPA 2.0 comes with a dedicated specification called JSR 317.

JPA 2.0 includes a number of additional new features:

**new mapping capabilities**

JPA 2.0 includes new mapping capabilities so that collections of embeddable objects and simple data types can be mapped in separate tables. In JPA 1.0, you could map only entity collections.

**extended mapping support**

JPA 2.0 supports extended mapping so that you can map keys, basic types, entities, and embeddable objects.

**standardized properties**

Properties in the persistence.xml file have been standardized, which improves the portability of custom applications.

**improved embedding of objects**

JPA 2.0 enables you to nest embeddable objects in other embeddable objects, and to create relational links to entities.

**pessimistic locking**

Pessimistic locking was introduced in JPA 2.0 to supplement the optimistic locking already supported in JPA 1.0. Pessimistic locking locks a resource when it's accessed by a transaction, whereas optimistic locking creates a copy of a resources state when the resource is first accessed.

**orphan removal**

JPA 2.0 enables you to remove orphaned records when the associated parent records have been removed.

**persistent record ordering**

You can use the @OrderColumn annotation to keep a record order persistent.

**Criteria API, and**

JPA 2.0 includes the new Criteria API, which enables you to construct queries based on objects, instead of using linear string-based methods.

**extended navigation syntax**

The . or dot navigation syntax has been extended to enable navigation through embedded embeddables and embeddables with existing relationships.

Question

Identify the correct statements about Java Persistence technology.

**Options:**

1. Java Persistence technology includes the JPA
2. Java Persistence technology uses a technology-specific query language
3. The Java Persistence API 2.0 was first released with JEE 5
4. The Java Persistence Query Language doesn't support the use of case statements

Answer

***Option 1:****Correct. Java Persistence technology includes the JPA, which enables you to perform create, read, update, and delete operations.*

***Option 2:****Correct. You use JPQL to retrieve information from the database.*

***Option 3:****Incorrect. JPA 1.0 was released with Java EE 5 and was updated to 2.0 for Java EE 6.*

***Option 4:****Incorrect. With the release of JPA 2.0, the JPQL also supports the use of case statements.*

**Correct answer(s):**

1. Java Persistence technology includes the JPA  
2. Java Persistence technology uses a technology-specific query language

Question

Identify features that were introduced with JPA 2.0.

**Options:**

1. The ability to remove orphaned records
2. The ability to navigate through related embedded objects
3. The ability to create linear string-based queries
4. The ability to create optimistic locks on database objects and resources

Answer

***Option 1:****Correct. In JPA 2.0, you can remove records that are orphaned when parent records are removed.*

***Option 2:****Correct. Extended navigation syntax enables you to use the . syntax to navigate through embedded objects and embedded objects with existing relationships.*

***Option 3:****Incorrect. The Criteria API included in JPA 2.0 now enables you to create queries based on objects.*

***Option 4:****Incorrect. JPA 2.0 enables you to create pessimistic locks on database objects and resources. JPA 1.0 introduced support for optimistic locks.*

**Correct answer(s):**

1. The ability to remove orphaned records  
2. The ability to navigate through related embedded objects

**2. Using JPA**

Java programming is all about objects, but when discussing persistence of objects in a relational database environment, these are referred to as *entities*. Objects are fleeting instances stored in memory, where they remain for a very short time.

An entity is an instance of an object that resides in memory, but can persistently reflect the database contents. This persistence is achieved through persistent fields or properties, which use object/relational mapping annotations to map to database contents.

An entity can usually be depicted as a table in a database. An *entity instance* is any one record in the table. During application development, entities are represented by entity classes, and can also inherit helper classes.

Entities also support database inheritance and relationships, and can be mapped to a database. You use JPQL to manipulate the data.

JPA uses ORM principles to establish a connection between objects and tables by mapping classes, objects, and attributes to their matching rows and columns in a relational database.

JPA provides some default mapping rules, and works on the *configuration by exception* concept. This means that the default mapping rules will be followed, except when specific rules are defined that override the default rules.

Metadata describes the mapping between objects in a relational database. You can use two formats to write metadata:

**annotations and**

Annotations are included in the javax.persistence package. You directly annotate the entity code to map to the relational database.

**XML descriptors**

XML descriptors can be used in conjunction with annotation but can also be used on their own. The XML mapping is stored in an external XML file that is deployed with the entities.

Consider a database containing a table of composers with examples of their compositions. You can create a class that reflects the entries in the table and you can add annotations to override some values.

Code

@Entity  
public class Composer {  
  
    @Id @GeneratedValue  
    private Long id;  
    private String name;  
    @Column(nullable = false)  
    private String composition;  
    private Float length;  
    @Column(length = 2000)  
    private String description;  
    private Integer tracks;  
  
    // Constructors, getters, setters  
}

For instance, the entity Id is derived from the @javax.persistence.id annotation and the value is generated by the @GenerateValue persistence provider. This corresponds with a table's primary key in a relational database.

Graphic

*The following section of code is indicated:  
   @Id @GeneratedValue  
   private Long id;  
   private String name;*

The @Column annotation is used to change the default mapping, which in this case is applied to the composition and description fields. The annotation specifies that composition can't be null, and that description has a maximum of 2,000 characters.

Graphic

*The following section of code is indicated:  
   @Column(nullable = false)  
   private String composition;  
   private Float length;  
   @Column(length = 2000)  
   private String description;  
   private Integer tracks;*

JPA creates an object-oriented separation from the traditional way of using primary and foreign keys to query relational data. The API is build around the entity manager, which enables you to manage the entities and perform create, read, update, and delete operations on the database using JPQL.

You can use this code to create an entity manager that persists a Composer entity.

Code

EntityManagerFactory entManF = Persistence.createEntityManagerFactory("chapter02PU");  
EntityManager entMan = entManF.createEntityManager();  
entMan.persist(composer);

Through the entity manager, you can also create queries for the database using JPQL, which is similar to SQL. For instance, you can use this query to return all entries in which the composer is Bach.

Code

SELECT c FROM Composer c WHERE c.name = 'Bach'

Query execution can be dynamic, static, or native SQL. Dynamic queries are created at runtime whereas static queries, also called named queries, are created when a program is being developed.

Static queries can be created using annotations or XML metadata. When you use an annotation, you can define a query as part of the entity definition, using the @NamedQuery annotation.

Code

@Entity  
@NamedQuery(name = "findComposer", query = "SELECT c FROM Composer c WHERE c.name = 'Bach'")  
  
public class Composer {  
    @Id @GeneratedValue  
    private Long id;  
    private String name;  
    @Column(nullable = false)  
    private String composition;  
    private Float length;  
    @Column(length = 2000)  
    private String description;  
    private Integer tracks;  
  
    // Constructors, getters, setters  
}

You can then run the createNamedQuery() method to execute the query and return a list of entities that match the search criteria.

Code

EntityManager.createNamedQuery()

Java EE Connector Architecture is the medium for bi-directional communication between the Java EE components and enterprise information systems, or EIS.

An enterprise information system can include a number of components, such as mainframe transactional processing, enterprise resource planning or ERP, and nonrelational databases.

Only one instance of a Connector is needed to connect the EIS to Java components, and connections can be made to components hosted on any platform. This portability is due to Connector specifications that extend across multiple platforms.

The Connector specification facilitates contract negotiation between the application server and the system-level software drivers, also called resource adapters, for EIS resources.

The resulting contracts also enable EIS components to be plugged into the application server, and vice versa.

Examples of contracts defined by the Java EE Connector Architecture 1.6 specification are Generic Work Context and Security Inflow.

A resource adapter can support three levels of transactions:

* LocalTransaction, which provides local resource manager transactions
* XATransaction, which supports the XA distributed transaction model and the JTA XATransaction interface, and
* NoTransaction, which provides no transaction support

Java Enterprise applications use a JDBC resource to connect to relational databases using the API. Each application in a domain uses a separate JDBC resource to connect to relational databases.

You use Java servlets, JavaServer Faces pages, and enterprise beans to provide transactional connectivity to JDBC resources.

Question

Java EE Connector Architecture facilitates connections between two entities. Identify the two entities.

**Options:**

1. Java EE Components
2. An EIS
3. The JDBC API
4. The JPA

Answer

***Option 1:****Correct. The Java EE Connector Architecture negotiates bi-directional communication between the Java EE Components and the EIS.*

***Option 2:****Correct. The Java EE Connector Architecture negotiates bi-directional communication between the EIS and Java EE Components.*

***Option 3:****Incorrect. The JDBC API is used by Java EE applications to access relational databases.*

***Option 4:****Incorrect. The JPA is used to interact with relational databases through entities.*

**Correct answer(s):**

1. Java EE Components  
2. An EIS

Question

You've created a class called Clients that will represent the entity you use to persist relational data. You've created a JPQL query and decide to execute it statically as part of the entity definition.  
  
Which annotation do you use to make this query accessible?

**Code**  
@Entity  
INSERT THE MISSING CODE(name = "findClient", query = "SELECT c FROM Clients c WHERE c.name = 'JWestlein'")  
  
public class Clients {  
    @Id @GeneratedValue  
    private Long id;  
    private String name;  
    @Column(nullable = false)  
    private String CustNum;  
    private Float contact;  
    @Column(length = 2000)  
    private String history;  
    private String email;  
}

**Options:**

1. @NamedQuery
2. @createNamedQuery
3. @newQuery

Answer

***Option 1:****Correct. You use the @NamedQuery annotation to add the query to the entity definition.*

***Option 2:****Incorrect. The createNamedQuery() method is used to execute the named query.*

***Option 3:****Incorrect. You use the @NamedQuery annotation to do this.*

**Correct answer(s):**

1. @NamedQuery

**3. Summary**

The JPA provides a way of persisting relational data, and is built around the concept of ORM. ORM establishes a link between objects and tables in a relational database. You use the JPQL to create queries to retrieve data from a database.  
  
Entities are used to persist objects, which temporarily reside in memory. JPA is based on the principle of configuration by exception, which means you only have to program the mapping rules that must override the defaults. To connect to an EIS, you employ Java EE Connector Architecture. To connect to a relational database, you use the JDBC API.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#top)

Understanding Java EE 6

Learning Objectives

*After completing this topic, you should be able to*

* *describe how Java EE 6 works*
* *determine which Java EE web component to use in a given scenario*
* *determine which web service to use in a given scenario*
* *determine when to use the different types of Enterprise JavaBeans*

**1. Exercise overview**

In this exercise, you're required to recognize how Java Enterprise Edition 6 – or Java EE 6 – works.

This involves the following tasks:

* identifying the characteristics of Java EE 6 and its components
* choosing a Java web component technology
* choosing a web service, and
* distinguishing between the Enterprise JavaBean, or EJB, types

**2. Identifying Java EE 6 characteristics**

You're using Java EE 6 to develop a series of new enterprise applications for your company.

Question

What are some of the features of Java Persistence technology?

**Options:**

1. Has its own query language
2. Uses an API to manipulate data
3. Uses the Criteria API to perform operations
4. Doesn't support case statements in queries
5. Retains orphaned records

Answer

***Option 1:****Correct. JPA 2.0 uses the Java Persistence Query Language, or JPQL to access data in a database.*

***Option 2:****Correct. You use the Java Persistence API, or JPA, to perform create, read, update, and delete operations on database information.*

***Option 3:****Correct. You can use the Java Persistence Criteria API to construct queries based on objects.*

***Option 4:****Incorrect. JPA 2.0 enables you to use case statements in the Java Persistence Query Language, or JPQL.*

***Option 5:****Incorrect. Deleting objects in the database creates orphaned records, which JPA 2.0 enables you to remove.*

**Correct answer(s):**

1. Has its own query language  
2. Uses an API to manipulate data  
3. Uses the Criteria API to perform operations

Question

Identify a key feature of an EJB.

**Options:**

1. JavaBeans run in the EJB container
2. Business logic is contained on the client side
3. Business logic is handled for you

Answer

***Option 1:****Correct. System-level services are provided by the EJB container enabling the developer to focus on the business logic of the bean.*

***Option 2:****Incorrect. The bean contains the business logic and as a result, creates thinner clients with less overhead.*

***Option 3:****Incorrect. System-level services in the EJB container are provided, however you need to handle the business logic.*

**Correct answer(s):**

1. JavaBeans run in the EJB container

Question

Which features are characteristics of web application components?

**Options:**

1. Manage HTTP requests received from web clients
2. Can contain EJB Lite
3. Used to create user interfaces for executing in web browsers
4. Process transactional business logic

Answer

***Option 1:****Correct. Web applications manage HTTP requests received from web clients, which are executed in web containers.*

***Option 2:****Correct. Web application components are able to contain EJB Lite since they were introduced in Java EE 6.*

***Option 3:****Incorrect. User interfaces that can be executed in web browsers are created by applets using the Swing API.*

***Option 4:****Incorrect. Enterprise applications contain EJB components that process transactional business logic.*

**Correct answer(s):**

1. Manage HTTP requests received from web clients  
2. Can contain EJB Lite

**3. Using Java EE 6 web components**

Question

You're writing a news delivery application that will enable users to filter the types of stories they receive on a news web site. The program will pull the information from a database and return it, before it's dynamically rendered in the user's browser.  
  
Identify the two Java technologies you can use to do this.

**Options:**

1. Java Server Pages, or JSP
2. Java Servlets
3. JavaServer Faces
4. Java Facelets

Answer

***Option 1:****Correct. You use JSP to generate web pages dynamically based on a client request, using data returned by Java Servlets.*

***Option 2:****Correct. You use Java Servlets to receive the requests for dynamic data from a client, and return this information to a JSP, which dynamically generates the web page.*

***Option 3:****Incorrect. You can use JavaServer Faces to extend an existing web infrastructure, and Facelets to provide the content in a visual interface.*

***Option 4:****Incorrect. You can use Java Facelets to present a front end to users using HTML templates, using data returned by JavaServer Faces.*

**Correct answer(s):**

1. Java Server Pages, or JSP  
2. Java Servlets

Question

You're contracted to a local sports team that wants its fans to be able to fill in membership application forms online. The team already has a web site and wants you to extend its functionality. The new season kicks off in three weeks and the team has asked you to have the system online before it starts.  
  
Which two Java technologies can you use to implement the application?

**Options:**

1. JavaServer Faces
2. Java Facelets
3. JSP
4. Java Servlets

Answer

***Option 1:****Correct. You can use JavaServer Faces to extend the existing web site by adding the new application pages and components.*

***Option 2:****Correct. You can use Java Facelets to finish the project quickly by using existing HTML templates, and also save time during the compiling and rendering.*

***Option 3:****Incorrect. You use JSP when you want to generate web pages based on dynamic data.*

***Option 4:****Incorrect. You use Java Servlets when you want clients to be able to request and receive data based on dynamic requests.*

**Correct answer(s):**

1. JavaServer Faces  
2. Java Facelets

**4. Choosing the correct web service**

Question

You work for a leasing agency that has a small web site, which displays all of its property listings. You want to create a web service that enables clients to browse through a list of only available properties.  
  
Which type of service can you use to do this?

**Options:**

1. RESTful web services
2. JAX-WS
3. Simple Object Access Protocol, or SOAP

Answer

***Option 1:****Correct. RESTful web services are best used for small-scale deployments that don't need extensive infrastructure, but are still scalable to facilitate a growing number of requests.*

***Option 2:****Incorrect. JAX-WS are best used in enterprise environments that have established Quality of Service, or QoS rules to adhere to.*

***Option 3:****Incorrect. SOAP is a protocol used by web services to exchange information between the client and server. It isn't a web service.*

**Correct answer(s):**

1. RESTful web services

Question

You've been hired by a large corporation that has offices spread across the world. Employees interact and share data frequently, and projects are often undertaken jointly. Strict security and privacy rules must be followed at all times.  
  
You've been asked to create a web service that will make it easier for employees to interact and share work resources. Which type of web service is most suitable in this scenario?

**Options:**

1. RESTful web services
2. JAX-WS
3. SOAP

Answer

***Option 1:****Incorrect. You use RESTful web services when you want to create a simple service for a small organization, but still want to ensure the service can cope with a growing numbers of requests.*

***Option 2:****Correct. JAX-WS are best suited for enterprise-sized organizations where reliable service and tightly controlled security are important.*

***Option 3:****Incorrect. SOAP is used to exchange information. It's a protocol rather than a web service.*

**Correct answer(s):**

2. JAX-WS

**5. Distinguishing JavaBean types**

Question

You're writing an application that will serve a large number of clients simultaneously, which also needs to be scalable. Which type of bean should you use?

**Options:**

1. Session bean
2. Message-driven bean

Answer

***Option 1:****Correct. You need to use a session bean because this type of bean supports application scalability and can handle multiple clients at once.*

***Option 2:****Incorrect. You need to use a session bean in this case. Message-driven bens don't provide scalability and can't handle multiple clients simultaneously.*

**Correct answer(s):**

1. Session bean

Question

Which are features of a message-driven bean?

**Options:**

1. Invoked when a client sends a message to the object for which the bean acts as a listener
2. Instance variables represent the state for handling messages
3. Client state is reset when the called method is executed
4. Contains a state-specific instance variable for as long as its methods are invoked

Answer

***Option 1:****Correct. Message-driven beans are invoked when a message is received on an object to which the bean listens.*

***Option 2:****Correct. Message-driven beans use instance variables to represent state.*

***Option 3:****Incorrect. The client state is reset after a method is executed for session beans.*

***Option 4:****Incorrect. Session beans use instance variables to persist a state.*

**Correct answer(s):**

1. Invoked when a client sends a message to the object for which the bean acts as a listener  
2. Instance variables represent the state for handling messages

Question

You're creating an application that will use a bean to track user responses during a survey. What kind of bean should you use?

**Options:**

1. Stateful session bean
2. Stateless session bean
3. Singleton session bean
4. Message-driven bean

Answer

***Option 1:****Correct. You use a stateful session bean in this case to maintain state between client responses.*

***Option 2:****Incorrect. A stateless bean can't maintain state between user responses during the survey.*

***Option 3:****Incorrect. A singleton bean can maintain only a single state for the entire duration of an application. You need to maintain state between client responses.*

***Option 4:****Incorrect. You need to use a session bean to maintain state between user responses.*

**Correct answer(s):**

1. Stateful session bean

Question

You're creating a bean that can convert plain text documents that it receives into configuration profiles for an application. What type of bean should you use in this case?

**Options:**

1. Message-driven bean
2. Stateful session bean
3. Stateless session bean
4. Singleton session bean

Answer

***Option 1:****Correct. You need the bean to be instantiated when a configuration message is received, so you have to use a message-driven bean.*

***Option 2:****Incorrect. You need to invoke the bean by sending a message rather than by invoking business logic methods.*

***Option 3:****Incorrect. You want the bean to be instantiated automatically when a message is received.*

***Option 4:****Incorrect. You need the bean to respond to a message that is received rather than being initiated when the application starts up.*

**Correct answer(s):**

1. Message-driven bean

The workings of JAVA EE 6 have been described, the best Java EE components to use in a given scenario have been identified, the best web service to use in a given scenario has been determined, and the use cases for the different types of Enterprise JavaBeans have been recognized.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#top)

The Java EE 6 Environment and Application Components

Learning Objective

*After completing this topic, you should be able to*

* *describe the key tasks involved in the preparation of a Java EE 6 Environment*

**1. Preparing a Java EE 6 environment**

Before you can start developing an application that uses the Java Enterprise Edition, or Java EE, platform, you need to set up a development environment.

A Java EE development environment has two main components:

**the Java EE Software Development Kit , or SDK and**

The Java Development Kit, or JDK, enables you to code and test Java applications. It contains the Java Virtual Machine and the core Java APIs optimized for use on workstations.  
  
The Java EE SDK provides additional libraries that you use with the JDK to develop server-based applications for the Java EE platform. For example, the SDK includes Enterprise Java Beans – or EJB, the Java Persistence API – or JPA, JavaServer Pages – or JSP, and various web services.  
  
You can download a version of the SDK that includes the JDK from this URL:  
  
http://www.oracle.com/technetwork/java/javaee/  
downloads/index.html

**an application server**

A Java EE application server is software running on a server that provides data from a Java EE application to clients. You need to set up this type of server to support the large scalable network applications that Java EE development is intended for.  
  
The Java EE 6 SDK includes GlassFish Server, which is an open-source application server for enterprises.  
  
You can download the latest version of GlassFish Server from this URL:  
  
http://glassfish.java.net/

You can use an integrated development environment, or IDE, to make it easier to create and manage the code of your application.   
  
NetBeans IDE is a free, open-source environment for developing applications in various programming languages. The Java EE version of NetBeans IDE is a complete package that includes the environment as well as the SDK and GlassFish Server.

Note

*You can download the NetBeans IDE from this URL:  
  
http://www.netbeans.org/downloads/index.html*

NetBeans IDE includes tools for monitoring the CPU and memory usage of an application. Using the IDE's debugger, you can track the execution of your application right down to stepping through the code line by line. The IDE also provides a visual way of building graphical user interfaces, or GUIs, for Java applications.  
  
There are hundreds of plugins you can download to add more services and features to NetBeans IDE.

You can integrate the NetBeans IDE with the GlassFish Server for Java EE application development. To do this, you need to register the GlassFish application server you've configured with the IDE.

In NetBeans IDE, you perform four steps to register a GlassFish application server:

* select **Servers** from the NetBeans IDE 7.0.1 **Tools** menu   
  *To do this, you select Tools - Servers.*
* specify GlassFish as the type of server   
  *On the Choose Server page of the Add Server Instance Wizard, you select GlassFish Server 3.x from the Server list box.*
* provide the location of the folder where you installed the application server, and   
  *On the Server Location page of the Add Server Instance Wizard, C:\glassfish3 is entered in the Installed Location text box.*
* specify the domain in which the application server operates   
  *On the Domain Location page of the Add Server Instance Wizard, the Register Local Domain radio button is selected and domain1 is specified in the Domain drop-down list box.*

The wizard for creating a new NetBeans IDE project guides you through the process of selecting the type of Java EE project you want to create, specifying the name of the project and where to save the project files, and choosing the application server to use.

Graphic

*The New Enterprise Application Wizard lists three steps:   
  
1. Choose Project  
2. Name and Location  
3. Server and Settings*

The main interface has three tabs: Projects, Files, and Services. These make it easy to manage the projects, files, and services you use during application development.

Graphic

*The three tabs are Projects, Files, and Services.*

On the Services tabbed page, you can access the Update Center to check for any updates to the SDK and to download add-ons for GlassFish Server.

Graphic

*To access the Update Center, you expand the Servers node, right-click the GlassFish Server 3.x node, and select View Update Center.*

Question

Identify the tasks involved in preparing a Java EE 6 environment.

**Options:**

1. Download the Java EE 6 SDK
2. Configure an IDE, such as GlassFish
3. Install an application server
4. Register the application server with the JDK

Answer

***Option 1:****Correct. To code Java EE 6 applications, you need to get the free SDK from the Internet. The JDK is bundled with the SDK and includes the Java Virtual Machine and workstation APIs.*

***Option 2:****Incorrect. GlassFish is an example of an application server and not an IDE. For Java EE 6 development, you can use NetBeans IDE instead.*

***Option 3:****Correct. For Java EE 6 development, you need to configure an application server, such as GlassFish. This server is responsible for managing data exchange between the Java EE application on the server and the clients.*

***Option 4:****Incorrect. You register the application server with NetBeans IDE, not the JDK. The JDK is for coding Java applications.*

**Correct answer(s):**

1. Download the Java EE 6 SDK  
3. Install an application server

**2. Summary**

You need to prepare a development environment for the Java EE applications you want to code. This requires the Java EE SDK and an application server to provide data from the application to clients. GlassFish Server is an example of a free application server and is included with the SDK. You can use an IDE to manage the coding. NetBeans IDE is a free, full-featured IDE and includes the SDK and GlassFish Server.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#top)

Stepping Through a Basic Java EE 6 Application

Learning Objectives

*After completing this topic, you should be able to*

* *match the component with how it will be used in a Java EE 6 application*
* *describe the creation of a basic Java EE 6 application*

**1. Components in a Java EE 6 application**

A basic Java Enterprise Edition, or Java EE, 6 application has four main components that interact with one another to make the application work.  
  
Consider a sample application called numberproject, which prompts the user to enter their favorite number and then compares it to the developer's favorite number. The application works out the difference between the two numbers, as well as the average difference among all user-submitted numbers.

The four main components of a basic Java EE 6 application are

**a web application**

The web application, numberproject, was developed with JavaServer Faces Facelets technology.   
  
It accesses the FavoriteNumberResource service to retrieve the developer's favorite number, and provides the user with an input screen. When the user enters their favorite number, the application uses the FavoriteNumberBean EJB to calculate the difference between the numbers. It also computes the average difference of all user numbers that have been entered. This is then displayed on screen.  
*The NetBeans IDE contains the numberproject project. There are four panes: Project displays the project files, Navigator helps you navigate through chunks of code, the Tasks and Output pane displays running tasks and generated output, and the main pane displays the code in the open project files.*

**a web service**

The web service is of the JAX-RS RESTful type and is called FavoriteNumberResource. It creates a new instance of FavouriteNumberResource and then returns a string representation of the developer's favorite number.  
RESTful web services are applications based on the web that access resources using the HTTP protocol. You can use a resource by identifying it using its URI and calling methods such as POST and GET.  
*The favorite-number project has been opened and the nodes expanded to show the FavoriteNumberResource object, which is nested in RESTful Web Services.*

**an Enterprise JavaBean, or EJB, and**

FavoriteNumberBean is a local, no-interface view, stateless session bean. It calculates the difference between the numbers and persists the data in an entity using the Java Persistence API.  
  
A stateless session bean used in enterprise applications does not create a conversation thread between itself and the client. In this case, the client makes a single request and previous requests or states are not maintained.  
*The numberproject project folders have been expanded and the FavoriteNumberBean EJB is located in the Enterprise Beans folder.*

**a Java Persistence API, or JPA**

The JPA entity for this application is called NumberProjectUser.java. When it's called by the EJB, it saves the user's number and the difference between the two numbers*.* This data is stored in a table in a Java database, and you use the business methods that form part of FavoriteNumberBean.java to perform the calculations.   
  
The JPA enables you to interact with data in a database through virtual objects called entities. An entity represents a table or a set of relational data in a database, and defines properties and constructors. Data can be created, read, updated, and deleted using the JPA Entity Manager, which has a number of methods. You use the Java Persistence Query Language, or JPQL, to query data in the database.  
*The Source Packages folder contains three packages – numberproject.ejb, numberproject.entity, and numberproject.web. The package numberproject.entity contains the NumberProjectUser.java file.*

The web application is built from

**an XHTML page**

The greeting.xhtml page is built from JavaServer Faces Facelets tag libraries and asks the user to enter their favorite number. It also reads the input.  
  
The messages displayed in this file and the response.xhtml file aren't actually coded into these files. Instead they use properties to get string values from the WebMessages.properties file.  
*The numberproject Web Pages folder contains two folders – WEB-INF and resources – and two files – greeting.xhtml and response.xhtml.*

**a Facelets-enabled XHTML page**

The response.xhtml page supports Facelets and tells the user how big the difference is between the numbers, and the average difference for all users who have entered their favorite numbers.

**a JavaServer Faces managed bean**

FavoriteNumber.java is a backing bean that acts as temporary storage for the data. It works with the web service and the EJB, and returns the result of the calculations to the browser by calling the response.xhtml page.  
*The numberproject.web folder contains FavoriteNumber.java and WebMessages.properties.*

**an enterprise bean**

The FavoriteNumberBean.java enterprise bean is packaged with the numberproject application and performs the calculations.  
*numberproject.ejb contains FavoriteNumberBean.java.*

**Java properties files, and**

WebMessages.properties is the Java properties file and is used to store the localized strings that appear on the response.xhtml and greeting.xhtml files. This file contains the English version of the strings – you can create more properties files for other languages.  
*The numberproject.web package contains FavoriteNumber.java and WebMessages.properties.*

**the web application's deployment descriptor**

The web.xml file is the application's deployment descriptor. It configures certain parts of the web application during installation. In this example it maps the application's FacesServlet instance. This is used to accept incoming requests and incorporate them into the application life cycle, where they're processed. It also initializes resources and specifies the welcome file for the application, which in this case is greeting.xhtml.  
*The Configuration Files folder contains five files – MANIFEST.MF, beans.xml, faces-config.xml, persistence.xml, and web.xml.*

The sample application is made up of a number of types of tier components:

Graphic

*A sample application consists of three layers. The client layer consists of Application client and optional JavaBean components, and Web browser, web pages, applets, and optional JavaBean components. The Java EE server layer consists of JavaBean components and Web pages servlets in the web tier, and Java persistent entities, Session beans and Message-driven beans in the business tier. The EIS layer consists of database and legacy systems. The client tier can interact directly with the database and legacy systems tier via the Application client and optional JavaBean components container. Alternatively the Web browser, web pages, applets, and optional JavaBean components can connect to the web tier via web page servlets, which in turn connect to the business tier via optional JavaBean components in the Java EE server tier. Finally, the business tier can also connect to the database and legacy systems.*

**one client-tier component**

The client-tier component is the user's web browser. The user accesses the rest of the application through the web tier.

**one web-tier component**

The web-tier component is the numberproject web application, also considered a web client.

**three business-tier components, and**

The three business-tier components are the FavoriteNumberResource web service, the FavoriteNumberBean enterprise bean, and the NumberProjectUser entity.

**one enterprise information system, or EIS, tier**

The EIS tier component is the data in the Java database that's connected by the JDBC API.

Question

Match each Java EE 6 component with a description of its function in an application.

**Options:**

1. Web service
2. EJB
3. JPA
4. Web application

**Targets:**

1. Reads and returns a value to the main program
2. Receives data and performs business operations on it
3. Receives data from another component and stores it in an entity
4. Invokes a component to retrieve data and forwards it to the component that performs the business processes

Answer

*The web service is called by the web application and returns data to be used for business processes.*

*The web application invokes the EJB using the web service data, which then performs business processes on it. The new data is then stored through a call to the JPA, and returned to the web application.*

*The JPA is called by the EJB which sends data in the form of parameters. This is then stored in an entity which in turn, is stored in a table in the Java database.*

*The web application invokes the web service to retrieve the data, then sends this data to the EJB where business processes are performed and resulting data is returned.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

**2. Creating a basic Java EE 6 application**

**Drill Down Home Page**

It's best practice to create a web service, the JPA entity, and an enterprise bean prior to creating a web client itself.

**Page 1 of 3: Creating the web service**

In Java EE 6, you can use two APIs to implement web services. For a SOAP-based web service, you use the JAX-WS API. For a RESTful web service, you use the JAX-RS API. Both have advantages but RESTful web services are simpler than SOAP and allow for faster transfers.

The main difference lies in how data is transported. A RESTful service uses HTTP directly as the transport protocol for data, whereas SOAP uses XML-formatted documents to transport data via HTTP.

**Page 2 of 3: Creating the web service**

By design, web services aren't dependent on any specific client – different types of client applications on multiple platforms can interact with RESTful web services. Connections are established through HTTP-based requests and responses, and client and server don't need to exchange implementation details. This is referred to as *loose coupling*.

Loose coupling enables you to develop a web service module as part of a different application module that will contain the rest of a program. It enables you to deploy the service on a different computer from the one running the client application, which is how web services are used.

**Page 3 of 3: Creating the web service**

You can create the FavoriteNumberResource web service by using the JAX-RS plug-in for NetBeans. This also generates the resource class you'll use. You use the javax.ws.rs.Path import statement to give the class access to the URL suffix supplied with the client request. This service uses a single method named getText(), and for this you need to use the javax.ws.rs.GET and javax.ws.rs.Produces import statements.

Graphic

*The FavoriteNumber project is open in the NetBeans interface. In the RESTful Web Services folder FavouriteNumberResource [favouriteNumber] has been selected and the FavouriteNumberResource.java file is open in the editor pane.*

Code

package mynumbers.webservice;  
import javax.ws.rs.core.Context;  
import javax.ws.rs.core.UriInfo;  
import javax.ws.rs.Path;  
import javax.ws.rs.GET;  
import javax.ws.rs.Produces;

Since the RESTful web service responds to HTTP requests, you use the @GET annotation to mark the method as an HTTP responder. You also use the @Produces annotation to set the getText() client response to a plain text-type MIME format.

Code

...  
import javax.ws.rs.Produces;  
@Path("favoriteNumber")  
public class FavoriteNumberResource {  
    @Context  
    private UriInfo context;  
    public FavoriteNumberResource() {  
    }  
    @GET  
    @Produces("text/plain")  
    public String getText() {  
        return "" + 10;  
   }  
}

**Page 1 of 3: Creating the JPA entity**

The NumberProjectUser JPA entity stores the current user's favorite number and the difference between the numbers. This entity class also includes the JPQL query to calculate the average difference from all user values. You start creating this class by selecting **New File** from the **File** menu, and completing the wizard using the Persistence category and Entity Class file type.

Graphic

*The New File Wizard is open on the Choose File Type page, which contains a Project drop-down list box set to numberproject, as well as Categories and File Type panes.*

Once you've done this, NetBeans generates a skeleton program and stores the NumberProjectUser.java file in the numberproject.entity package. The file also opens enabling you to add the entity properties.

Code

/\*  
 \* To change this template, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/  
package numberproject.entity;  
import java.io.Serializable;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
/\*\*  
 \*  
 \* @author Jonathan  
 \*/  
@Entity  
public class NumberProjectUser implements Serializable {  
    private static final long *serialVersionUID* = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.*AUTO*)  
    private Long id;

**Page 2 of 3: Creating the JPA entity**

To save the user's favorite number and the difference, you create two new properties for the entities. Because these are whole numbers you use the int variable, and add the declarations under the id declaration.

Code

public class NumberProjectUser implements Serializable {  
    private static final long *serialVersionUID* = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.*AUTO*)  
    private Long id;  
    protected int number;  
    protected int difference;

NetBeans automatically adds the getId() and setId() methods to the generated skeleton. You now add the get and set methods for the properties you just created.

Code

    public Long getId() {  
        return id;  
    }  
    public void setId(Long id) {  
        this.id = id;  
    }  
    public int getNumber() {  
        return number;  
    }  
    public void setNumber(int number) {  
        this.number = number;  
    }  
    public int getDifference() {  
        return difference;  
    }  
    public void setDifference(int difference) {  
        this.difference = difference;  
   }

You add the two constructors for NumberProjectUser, the first taking no parameters and the second one for number and one for difference. You add this code after the entity properties declaration but before the get and set methods.

Code

    public NumberProjectUser() {  
    }  
    public NumberProjectUser(int number, int difference) {  
        this.setNumber(number);  
        this.setDifference(difference);  
    }  
    public Long getId() {  
        return id;  
    }  
    public void setId(Long id) {  
        this.id = id;  
    }  
    public int getNumber() {  
        return number;  
    }  
    public void setNumber(int number) {  
        this.number = number;  
   }

**Page 3 of 3: Creating the JPA entity**

To complete the NumberProjectUser JPA, you add the JPQL named query to the line just after the @Entity annotation. You use the @NamedQuery annotation and the JPQL SELECT statement with the AVG operator.

Code

@Entity  
@NamedQuery(name = "findAverageDifferenceOfAllNumberProjectUsers",  
query = "SELECT AVG(u.difference) FROM NumberProjectUser u")  
public class NumberProjectUser implements Serializable {  
    private static final long *serialVersionUID* = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.*AUTO*)  
    private Long id;  
    protected int number;  
    protected int difference;  
    public NumberProjectUser() {  
    }  
    public NumberProjectUser(int number, int difference) {  
        this.setNumber(number);  
        this.setDifference(difference);  
    }  
    public Long getId() {  
        return id;  
   }

**Page 1 of 3: Creating the enterprise bean**

To create an EJB, you select **New File** from the **File** menu, and select the **Session Bean** file type from the **Enterprise JavaBean** category. You then enter the name of the EJB and the package and ensure **Stateless** is selected.

Graphic

*The New Session Bean dialog window is open at the second step named Name and Location. There are a few input fields – the EJB Name is FavoriteNumberBean, Project is numberproject, Location is Source Packages, and Package is numberproject.ejb. The Session Type section contains three radio buttons – Stateless which is selected, Stateful, and Singleton. The Create Interface section has two checkboxes called Local and Remote. Both are deselected.*

NetBeans generates a basic structure for the EJB.

Code

/\*  
 \* To change this template, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/  
package numberproj.ejb;  
  
import javax.ejb.Stateless;  
import javax.ejb.LocalBean;  
  
/\*\*  
 \*  
 \* @author Jonathan  
 \*/  
@Stateless  
@LocalBean  
public class FavoriteNumberBeanDefault {  
  
    // Add business logic below. (Right-click in editor and choose  
    // "Insert Code > Add Business Method")  
      
}

You can now add the logger instance, which will log events, to the code. You can also include the persistence declaration and link the EJB to the NumberProjectUser JPA.

Code

package numberproj.ejb;  
  
import javax.ejb.Stateless;  
import javax.ejb.LocalBean;  
import java.util.logging.Logger;  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
import numberproject.entity.NumberProjectUser;

**Page 2 of 3: Creating the enterprise bean**

You now add the declaration for the event logger for the session bean. You also add the business logic the EJB will use to perform the calculations. First you create a new EntityManager instance using the @PersistenceContext annotation.

Code

package numberproj.ejb;  
import javax.ejb.Stateless;  
import javax.ejb.LocalBean;  
import java.util.logging.Logger;  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
import numberproject.entity.NumberProjectUser;  
@Stateless  
@LocalBean  
public class FavoriteNumberBean {  
    @PersistenceContext  
    private EntityManager em;  
    private static final Logger *logger* =  
        Logger.*getLogger*("numberproject.ejb.FavoriteNumberBean");

Next you add the method to calculate the average difference between the developer's number and all users who have entered theirs. In order to do this you have to call the findAverageDifferenceOfAllNumberProjectUsers query included in NumberProjectUser. The Entity Manager's createNamedQuery() method runs the query and the getSingleResult() method receives the number. The result and a message are written to the event logger.

Code

    public Double getAverageDifference() {  
        Double avgDiff = (Double)  
em.createNamedQuery("findAverageDifferenceOfAllNumberProjectUsers")  
.getSingleResult();  
        *logger*.info("Average difference is: "   
+ avgDiff);  
        return avgDiff;  
   }

**Page 3 of 3: Creating the enterprise bean**

The getDifference() method calculates the difference between the user's and the developer's favorite numbers. The method first checks which number is larger and then subtracts the two, before logging the result to the event logger. It also creates a NumberProjectUser entity and saves the new user's number in the Java DB using the persist method. It also returns the difference to the calling method.

Code

public int getDifference(int number) {  
    int difference;  
    int favoriteNumber = 10;  
    if (favoriteNumber > number) {  
        difference = favoriteNumber - number;  
    } else {  
        difference = number - favoriteNumber;  
    }  
    *logger*.info("The difference is: " + difference);  
    if (difference == 0) {  
        *logger*.info("The user has the same favorite number!");  
    }  
    NumberProjectUser user = new NumberProjectUser(number, difference);  
    em.persist(user);  
    return difference;  
   }

**3. Creating the web client**

The web client brings the components of the application together. To create the client, you need to add a few new files. You can start by adding the Resource Bundle by right-clicking **numberproject**and selecting **New**- **Other**.

You then navigate to the properties file type.

And you specify a name and location for the file.

NetBeans then automatically creates a blank file for you to use. This file can also be used for localization purposes, in which case you would create a separate file for each of the languages you want to support.

JavaServer Faces applications don't normally require a deployment descriptor file. But when you want to use features like the resource bundle, which can be configured only in this file, you have to include it. To do this, you select the **JSF Faces Configuration** file type from the **JavaServer Faces** category.

Graphic

*The Choose File Type page of the New File Wizard is open.*

You then add this XML code to the file. The base-name element of resource-bundle specifies the bundle's fully qualified class name. The var element provides a way for the XHTML pages to reference the resource bundle. The locale-config element identifies the localized files available to the bundle.

Code

    <application>  
        <resource-bundle>  
            <base-name>numberproject.web.WebMessages</base-name>  
            <var>bundle</var>  
        </resource-bundle>  
        <locale-config>  
            <default-locale>en</default-locale>  
        </locale-config>  
    </application>  
</faces-config>

You also create a backing bean for the client, which provides temporary storage for data on a JSF page. You add this bean by adding a new file to the project, and selecting the **JSF Managed Bean** type in the JavaServer Faces category. You save it in the .web package of your application.

Graphic

*The New File dialog window is open.*

You now edit the code skeleton to bind it with the application. First you change the value of the @Named annotation to match the name of the bean. You then import the FavouriteNumberBean package and use the @EJB annotation to reference it.

Code

package numberproject.web;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import javax.ejb.EJB;  
import javax.inject.Named;  
import javax.enterprise.context.SessionScoped;  
import java.io.Serializable;  
import java.net.HttpURLConnection;  
import java.net.MalformedURLException;  
import java.net.ProtocolException;  
import java.net.URL;  
import java.util.logging.Logger;  
import javax.validation.constraints.NotNull;  
import numberproject.ejb.FavoriteNumberBean;  
  
@Named(value = "FavoriteNumber")  
@SessionScoped  
public class FavoriteNumber implements Serializable {  
  
    @EJB  
    private FavoriteNumberBean favoriteNumberBean;

You continue to build the bean by adding the entity properties and initializing the values.

Code

    @EJB  
    private FavoriteNumberBean favoriteNumberBean;  
    protected int number;  
    @NotNull  
    protected int yourFavoriteNumber;  
    protected int diff;  
    protected Double averageDifference;  
    private static final Logger *logger* = Logger.*getLogger*("numberproject.web.FavoriteNumber");  
  
    public FavoriteNumber() {  
        number = 1;  
        yourFavoriteNumber = 1;  
        diff = 1;  
        averageDifference = 1.0;  
   }

You also add the getNumber() method to connect to the RESTful web service.

Code

    public int getNumber() {  
        // Use the java.net.\* APIs to access the Favorite Number RESTful web service  
        HttpURLConnection connection = null;  
        BufferedReader rd = null;  
        StringBuilder sb = null;  
        String line = null;  
        URL serverAddress = null;  
  
        try {  
            serverAddress = new URL(  
                    "http://localhost:8080/FavoriteNumberService/resources/favoriteNumber");  
            connection = (HttpURLConnection) serverAddress.openConnection();  
            connection.setRequestMethod("GET");  
            connection.setDoOutput(true);  
            connection.setReadTimeout(10000);  
  
            connection.connect();  
  
            rd = new BufferedReader(  
                    new InputStreamReader(connection.getInputStream()));  
            sb = new StringBuilder();  
            while ((line = rd.readLine()) != null) {  
                sb.append(line);  
           }

You log errors using the logger.

Code

...  
            // Convert the response to an int  
            number = Integer.parseInt(sb.toString());  
        } catch (MalformedURLException e) {  
            *logger*.warning("A MalformedURLException occurred.");  
            e.printStackTrace();  
        } catch (ProtocolException e) {  
            *logger*.warning("A ProtocolException occurred.");  
            e.printStackTrace();  
        } catch (IOException e) {  
            *logger*.warning("An IOException occurred");  
            e.printStackTrace();  
        }  
  
        return number;  
   }

Next you add the appropriate get and set methods, and close off the bean using the processNumbers() method, which calls the methods in the EJB to process the numbers.

Code

    public void setNumber(int number) { this.number = number; }  
  
    public int getYourFavoriteNumber() { return yourFavoriteNumber; }  
  
    public void setYourFavoriteNumber(int yourFavoriteNumber) {  
        this.yourFavoriteNumber = yourFavoriteNumber; }  
  
    public int getDiff() { return diff; }  
  
    public void setDiff(int diff) { this.diff = diff; }  
  
    public Double getAverageDifference() { return averageDifference; }  
  
    public void setAverageDifference(Double averageDifference) {  
        this.averageDifference = averageDifference; }  
  
    public String processNumbers() {  
        this.setDiff(favoriteNumberBean.getDifference(yourFavoriteNumber));  
        *logger*.info("Positive number difference from my favorite number " + diff);  
  
        this.setAverageDifference(favoriteNumberBean.getAverageDifference());  
        *logger*.info("averageDifference " + averageDifference);  
        return "/response.xhtml";  
   }  
}

The Facelets client consists of three components:

* a resource library   
  *The Web Pages node has been indicated. It consists of two folders, WEB\_INF and resources.*
* a composite component, and   
  *The resources folder is expanded and indicated. It consists of the file inputNumber.xhtml.*
* two XHTML files   
  *Two files, greeting.xhtml and response.xhtml, that are indicated.*

A JSF resource library brings together the user-created components used in a web application. You use a resource identifier, which points to a specific resource in a web application, to identify a resource library. Resources are saved in either the web application's root or in its classpath.

Graphic

*The numberproject project is open and the Web Pages node is expanded.*

Resources in the web application's root are saved in the Web Pages folder structure in the project.

Syntax

resources/*resource-identifier*

Resources in the web application's classpath should be stored in this project folder. In this example, the resources are stored in a components sub-folder, which contains the inputNumber.xhtml resource.

Syntax

META-INF/resources/*resource-identifier*

A composite component defines the JSF and Facelets components located in the resource. The inputNumber.xhtml resource contains the tags needed to read the user input and includes interface and implementation definitions.

Code

<?xml version='1.0' encoding='UTF-8' ?>  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:cc="http://java.sun.com/jsf/composite"  
      xmlns:h="http://java.sun.com/jsf/html"  
      xmlns:f="http://java.sun.com/jsf/core">  
  
    <!-- INTERFACE -->  
    <cc:interface>  
        <cc:attribute name="number" />  
    </cc:interface>  
  
    <!-- IMPLEMENTATION -->  
    <cc:implementation>  
        <h:inputText value="#{cc.attrs.number}">  
        </h:inputText>  
    </cc:implementation>  
  
</html>

The interface definition makes the attributes available to pages that use the composite component.

Code

    <cc:interface>  
        <cc:attribute name="number" />  
    </cc:interface>

The tags nested in the implementation element specify the components that will be added to the pages that use the composite component. You can then use an inputText element to retrieve the attribute value for this page.

Code

    <cc:implementation>  
        <h:inputText value="#{cc.attrs.number}">  
        </h:inputText>  
    </cc:implementation>

The greeting.xhtml file reads the localized strings from the WebMessages.properties file.

Graphic

*The relevant code is:  
  
#{bundle.Welcome}  
#{bundle.MyFavoriteNumber}  
#{bundle.Instructions}  
#{bundle.YourFavoriteNumber}  
#{bundle.Pattern}  
and  
#{bundle.Submit}*

Code

...  
    <h:head>  
        <title>Number Project Greeting Page</title>  
    </h:head>  
    <h:body>  
        <h:form>  
            <h2>  
                <h:outputText value="#{bundle.Welcome}"/>  
            </h2>  
            <h:outputText value="#{bundle.MyFavoriteNumber} "/>  
            <h:outputText value="#{FavoriteNumber.number} "/>  
            <p/>  
            <h:outputText value="#{bundle.Instructions}"/>  
            <p/>  
            <h:outputText value="#{bundle.YourFavoriteNumber} "/>  
            <fc:inputNumber id="userNumber" number="#{FavoriteNumber.yourFavoriteNumber}" />  
            <h:outputText value=" #{bundle.Pattern}"/>  
            <p/>  
            <h:commandButton value="#{bundle.Submit}"   
                             action="#{FavoriteNumber.processNumbers}"/>  
            <p/>  
            <h:message for="userNumber" style="color:red"/>  
        </h:form>  
   </h:body>

The XHTML file also calls the EJB and the RESTful web service through the FavoriteNumber backing bean. It uses the inputNumber composite component to create the text field that reads the user's input.

Graphic

*The relevant code is:  
  
<h:outputText value="#{FavoriteNumber.number} "/>  
<fc:inputNumber id="userNumber" number="#{FavoriteNumber.yourFavoriteNumber}" />  
and<h:commandButton value="#{bundle.Submit}" action="#{FavoriteNumber.processNumbers}"/>*

Code

...  
    <h:head>  
        <title>Number Project Greeting Page</title>  
    </h:head>  
    <h:body>  
        <h:form>  
            <h2>  
                <h:outputText value="#{bundle.Welcome}"/>  
            </h2>  
            <h:outputText value="#{bundle.MyFavoriteNumber} "/>  
            <h:outputText value="#{FavoriteNumber.number} "/>  
            <p/>  
            <h:outputText value="#{bundle.Instructions}"/>  
            <p/>  
            <h:outputText value="#{bundle.YourFavoriteNumber} "/>  
            <fc:inputNumber id="userNumber" number="#{FavoriteNumber.yourFavoriteNumber}" />  
            <h:outputText value=" #{bundle.Pattern}"/>  
            <p/>  
            <h:commandButton value="#{bundle.Submit}"  
                             action="#{FavoriteNumber.processNumbers}"/>  
            <p/>  
            <h:message for="userNumber" style="color:red"/>  
        </h:form>  
   </h:body>

The response.xhtml file displays the difference between the numbers – the strings used depend on the outcome of the calculations. It's based on the conditions specified by the rendered attribute, which uses the == or equal to, lt or less than, and gt or greater than operators.

Code

...      
    <h:head>  
        <title>Response Page</title>  
    </h:head>  
    <h:body>  
        <h:form>  
            <h:outputText value="#{bundle.YourFavorite} "/>  
            <h:outputText value="#{bundle.Same}" rendered="#{FavoriteNumber.diff == 0}"/>  
            <h:outputText value="#{FavoriteNumber.diff}" rendered="#{FavoriteNumber.diff gt 0}"/>  
            <h:outputText value="#{bundle.Number}" rendered="#{FavoriteNumber.diff == 1}"/>  
            <h:outputText value="#{bundle.Numbers}" rendered="#{FavoriteNumber.diff gt 1}"/>  
            <h:outputText value="#{bundle.Difference}" rendered="#{FavoriteNumber.diff gt 0}"/>  
            <p/>  
            <h:outputText   
                value="#{bundle.Average} #{FavoriteNumber.averageDifference}." />  
            <p/>  
            <h:commandButton id="back" value="#{bundle.Back}" action="greeting"/>  
        </h:form>  
   </h:body>

Question

Which component contains the business logic in a basic Java EE 6 web application?

**Options:**

1. EJB
2. Web client
3. JPA
4. Web service

Answer

***Option 1:****Correct. The EJB is called by the backing bean and receives the input for the business logic. It also interacts with the JPA to retrieve and persist the data.*

***Option 2:****Incorrect. The web client pulls all the components together and is built using JavaServer Faces Facelets, a backing bean, and a deployment descriptor.*

***Option 3:****Incorrect. The JPA uses a JPQL query to retrieve data from the Java DB and saves the new data back to it.*

***Option 4:****Incorrect. The web service provides the resources or hard-coded data to the application.*

**Correct answer(s):**

1. EJB

**4. Summary**

A basic Java EE 6 application can be built using four components – a web application, web service, EJB, and JPA. A web application is built using a combination of .xhtml pages, a JavaServer Faces and Enterprise-managed bean, a properties file, and a deployment descriptor. An EJB is used to perform the business operations, the JPA stores and retrieves data, and the web service provides dynamic access to resources or data. These components are arranged in tiers, each containing some of these components.  
  
You use annotations in the web service to enable it to read HTTP requests, and specify the type of response it should give. The JPA can contain get and set methods, constructors, and JPQL to query data from the Java DB. The EJB contains the business logic that perform the work of the application and returns the data to the calling components. These are all brought together in the web client, which also has a number of components.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#top)

How to Create a Basic Java EE 6 Application

Learning Objectives

*After completing this topic, you should be able to*

* *recognize how to prepare a Java EE 6 Environment*
* *describe the characteristics of the different components of a Java EE 6 application*

**1. Exercise overview**

In this exercise, you're required to recognize how to develop a Java Enterprise Edition, or Java EE, 6 application.

This involves the following tasks:

* determining how to prepare a Java EE 6 environment and
* recognizing how to build a Java EE 6 application using a number of components

**2. Preparing a Java EE 6 Environment**

You're the development team lead for a company that wants you to create a web application to provide services for its clients. You decide to use Java EE 6 to build the application and to employ a number of the available enterprise components. For example, users must be able to connect to a web service, and you want to be able to store the user's data in a relational database.

Question

You want to prepare the Java EE 6 environment that you and your team members will use to develop the application. Which tasks do you perform to set up the development environment?

**Options:**

1. Download and install the Java EE 6 software development kit, or SDK
2. Download and install the application server
3. Download and install the Java Virtual Machine, or JVM
4. Register the application server with the SDK

Answer

***Option 1:****Correct. You download and install the Java EE 6 SDK to code the program.*

***Option 2:****Correct. You download and install the application server to provide Java EE application data to clients.*

***Option 3:****Incorrect. The JVM forms part of the Java EE 6 SDK.*

***Option 4:****Incorrect. The application server should be registered with the integrated development environment, or IDE you've chosen.*

**Correct answer(s):**

1. Download and install the Java EE 6 software development kit, or SDK  
2. Download and install the application server

Question

You've downloaded and installed the Java EE 6 SDK and now want to provide the team with a means of coding the application. What is your next step?

**Options:**

1. Install an IDE, such as NetBeans
2. Register the application server with the IDE
3. Configure the GlassFish IDE
4. Download and integrate the core Java APIs

Answer

***Option 1:****Correct. You can freely download and install the NetBeans IDE to provide a development environment for your team.*

***Option 2:****Correct. You register the application server with the IDE to create network-based applications.*

***Option 3:****Incorrect. GlassFish is the application server bundled with the SDK.*

***Option 4:****Incorrect. The Java EE 6 SDK includes the core Java APIs that are optimized for use on workstations.*

**Correct answer(s):**

1. Install an IDE, such as NetBeans  
2. Register the application server with the IDE

**3. Java EE 6 application components**

Question

You want to develop a web service to provide client users with data. Which class do you import to enable the service to read the URL suffix that's sent with the client request?

**Code**  
import javax.ws.rs.core.Context;  
import javax.ws.rs.core.UriInfo;  
import INSERT THE MISSING CODE;

**Options:**

1. javax.ws.rs.Path
2. java.ws.rs.URLinfo
3. java.ws.rs.URL

Answer

***Option 1:****Correct. You import the javax.ws.rs.Path class to enable the service to read the URL suffix to which it will respond.*

***Option 2:****Incorrect. You import the javax.ws.rs.Path class to specify the URL suffix to which the service will respond.*

***Option 3:****Incorrect. You import the javax.ws.rs.Path class to enable the service to read the URL suffix to which the service will respond.*

**Correct answer(s):**

1. javax.ws.rs.Path

Question

You now want to build a Java Persistence API, or JPA with an entity called User. Each instance must have a unique generated id. Which annotation would you use to auto-generate the entity's id properties?

**Code**  
@Entity  
public class User {  
  
    @Id  
    INSERT THE MISSING CODE(strategy = GenerationType.*AUTO*)  
    private Long id;

**Options:**

1. @GeneratedValue
2. @GenerateID
3. @AutoGenerate

Answer

***Option 1:****Correct. You use the @GeneratedValue annotation to automatically generate an id for each entity instance.*

***Option 2:****Incorrect. You use the @GeneratedValue annotation to generate an id for each entity instance.*

***Option 3:****Incorrect. You use the @GeneratedValue annotation to generate unique identifiers for an entity in a JPA.*

**Correct answer(s):**

1. @GeneratedValue

Question

You want to include an Enterprise JavaBean, or EJB, in the application. Which functions will the EJB primarily be responsible for?

**Options:**

1. Performing business logic operations
2. Persisting the working-set data
3. Connecting to the web service
4. Temporarily storing user data

Answer

***Option 1:****Correct. The EJB's primary responsibility is performing business logic operations, which can include calculations or logical decision-making.*

***Option 2:****Correct. The EJB can interact with the JPA using the EntityManager class to store the working-set data in a table in the Java database.*

***Option 3:****Incorrect. The web service is invoked by the web client in response to a request.*

***Option 4:****Incorrect. In a Java EE 6 application you use a JavaServer Faces backing bean to provide temporary storage for the user data.*

**Correct answer(s):**

1. Performing business logic operations  
2. Persisting the working-set data

Question

Which components can you use to build the web client?

**Options:**

1. JavaServer Faces Facelets XHTML pages
2. JavaServer Faces-managed JavaBean
3. A properties file
4. A web service
5. JPA

Answer

***Option 1:****Correct. You use JavaServer Faces Facelets XHTML pages to create the web pages that users will interact with. These are easily extensible using tags to add components.*

***Option 2:****Correct. You use a JSF-managed JavaBean to provide temporary storage and interact with the EJB.*

***Option 3:****Correct. You can use a properties file to store localized string data to make your application portable across all platforms and in more than one language.*

***Option 4:****Incorrect. The web service doesn't form part of the web client and is a separate entity. It is accessed by clients using the client application.*

***Option 5:****Incorrect. You create the JPA separately from the client, which also enables it to be used by other applications that work with the same set of data.*

**Correct answer(s):**

1. JavaServer Faces Facelets XHTML pages  
2. JavaServer Faces-managed JavaBean  
3. A properties file

Steps for setting up a Java development environment and for developing a web application using a number of Java EE 6 components have been identified.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jteo_a01_it_enus/output/html/course_transcript.html#top)

© 2016 Skillsoft Ireland Limited