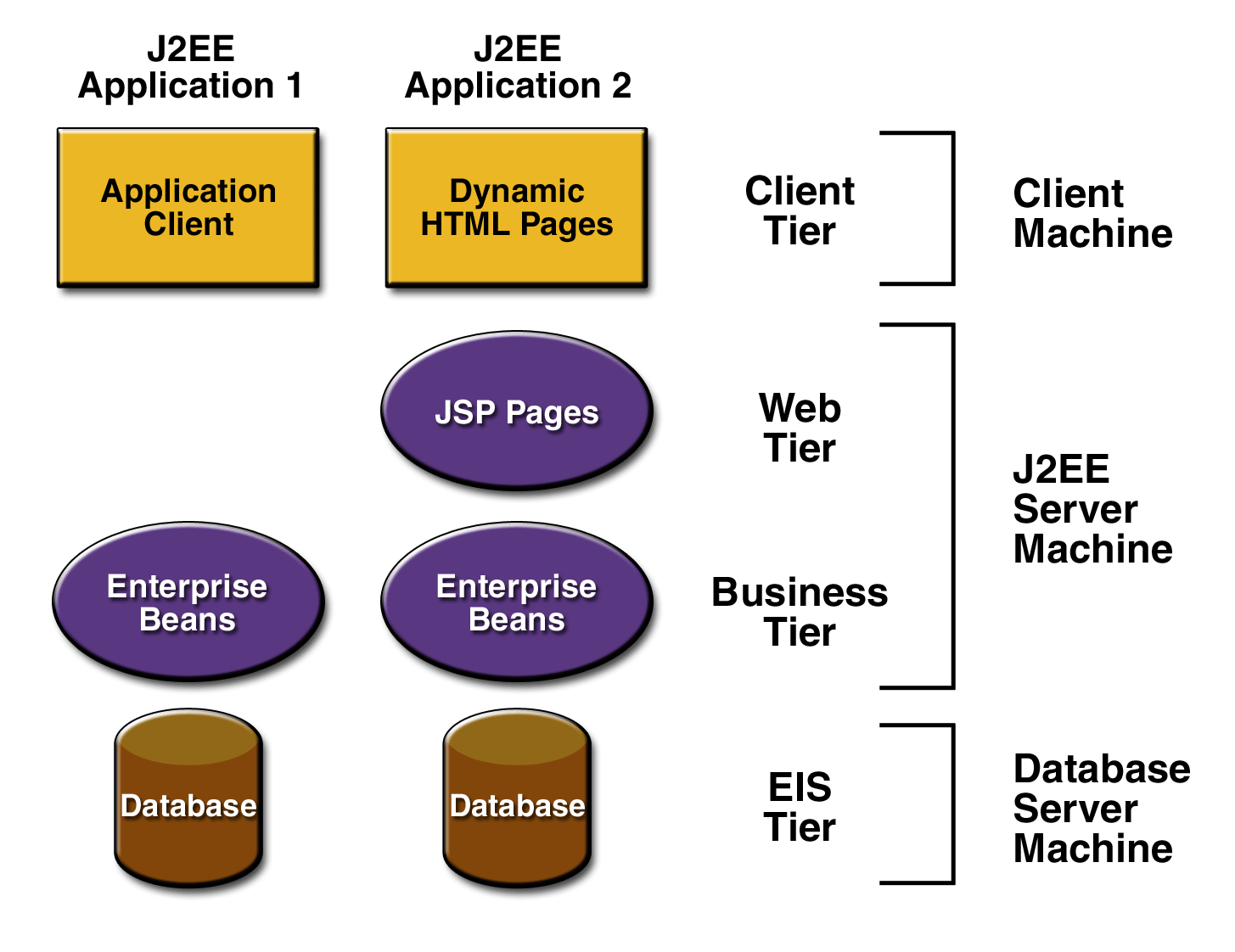
**Distributed Multitiered Applications**

The J2EE platform uses a multitiered distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multitiered J2EE environment to which the application component belongs. [Figure 1-1](http://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/Overview2.html#83323) shows two multitiered J2EE applications divided into the tiers described in the following list. The J2EE application parts shown in [Figure 1-1](http://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/Overview2.html#83323) are presented in [J2EE Components](http://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/Overview2.html#64857).

* Client-tier components run on the client machine.
* Web-tier components run on the J2EE server.
* Business-tier components run on the J2EE server.
* Enterprise information system (EIS)-tier software runs on the EIS server.

Although a J2EE application can consist of the three or four tiers shown in [Figure 1-1](http://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/Overview2.html#83323), J2EE multitiered applications are generally considered to be three-tiered applications because they are distributed over three different locations: client machines, the J2EE server machine, and the database or legacy machines at the back end. Three-tiered applications that run in this way extend the standard two-tiered client and server model by placing a multithreaded application server between the client application and back-end storage.



**Figure 1-1 Multitiered Applications**

**J2EE Components**

J2EE applications are made up of components. A *J2EE component* is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and that communicates with other components. The J2EE specification defines the following J2EE components:

* Application clients and applets are components that run on the client.
* Java Servlet and JavaServer Pageshttp://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/images/tm.gif (JSPhttp://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/images/tm.gif) technology components are Web components that run on the server.
* Enterprise JavaBeanshttp://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/images/tm.gif (EJBhttp://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/images/tm.gif) components (enterprise beans) are business components that run on the server.

J2EE components are written in the Java programming language and are compiled in the same way as any program in the language. The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server.

**J2EE Clients**

A J2EE client can be a Web client or an application client.

**Web Clients**

A Web client consists of two parts: dynamic Web pages containing various types of markup language (HTML, XML, and so on), which are generated by Web components running in the Web tier, and a Web browser, which renders the pages received from the server.

A Web client is sometimes called a *thin client*. Thin clients usually do not do things like query databases, execute complex business rules, or connect to legacy applications. When you use a thin client, heavyweight operations like these are off-loaded to enterprise beans executing on the J2EE server where they can leverage the security, speed, services, and reliability of J2EE server-side technologies.

**Applets**

A Web page received from the Web tier can include an embedded applet. An applet is a small client application written in the Java programming language that executes in the Java virtual machine installed in the Web browser. However, client systems will likely need the Java Plug-in and possibly a security policy file in order for the applet to successfully execute in the Web browser.

Web components are the preferred API for creating a Web client program because no plug-ins or security policy files are needed on the client systems. Also, Web components enable cleaner and more modular application design because they provide a way to separate applications programming from Web page design. Personnel involved in Web page design thus do not need to understand Java programming language syntax to do their jobs.

**Application Clients**

A J2EE application client runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language. It typically has a graphical user interface (GUI) created from Swing or Abstract Window Toolkit (AWT) APIs, but a command-line interface is certainly possible.

Application clients directly access enterprise beans running in the business tier. However, if application requirements warrant it, a J2EE application client can open an HTTP connection to establish communication with a servlet running in the Web tier.

**JavaBeanshttp://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/images/tm.gif Component Architecture**

The server and client tiers might also include components based on the JavaBeans component architecture (JavaBeans component) to manage the data flow between an application client or applet and components running on the J2EE server or between server components and a database. JavaBeans components are not considered J2EE components by the J2EE specification.

JavaBeans components have instance variables and get and set methods for accessing the data in the instance variables. JavaBeans components used in this way are typically simple in design and implementation, but should conform to the naming and design conventions outlined in the JavaBeans component architecture.

**J2EE Server Communications**

[Figure 1-2](http://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/Overview2.html" \l "79047) shows the various elements that can make up the client tier. The client communicates with the business tier running on the J2EE server either directly or, as in the case of a client running in a browser, by going through JSP pages or servlets running in the Web tier.

Your J2EE application uses a thin browser-based client or thick application client. In deciding which one to use, you should be aware of the trade-offs between keeping functionality on the client and close to the user (thick client) and off-loading as much functionality as possible to the server (thin client). The more functionality you off-load to the server, the easier it is to distribute, deploy, and manage the application; however, keeping more functionality on the client can make for a better perceived user experience.

**J2EE Containers**

Normally, thin-client multitiered applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details. The component-based and platform-independent J2EEarchitecture makes J2EE applications easy to write because business logic is organized into reusable components. In addition, the J2EE server provides underlying services in the form of a container for every component type. Because you do not have to develop these services yourself, you are free to concentrate on solving the business problem at hand.

**Container Services**

*Containers* are the interface between a component and the low-level platform-specific functionality that supports the component. Before a Web, enterprise bean, or application client component can be executed, it must be assembled into a J2EE application and deployed into its container.

The assembly process involves specifying container settings for each component in the J2EE application and for the J2EE application itself. Container settings customize the underlying support provided by the J2EE server, which includes services such as security, transaction management, Java Naming and Directory Interfacehttp://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/images/tm.gif (JNDI) lookups, and remote connectivity. Here are some of the highlights:

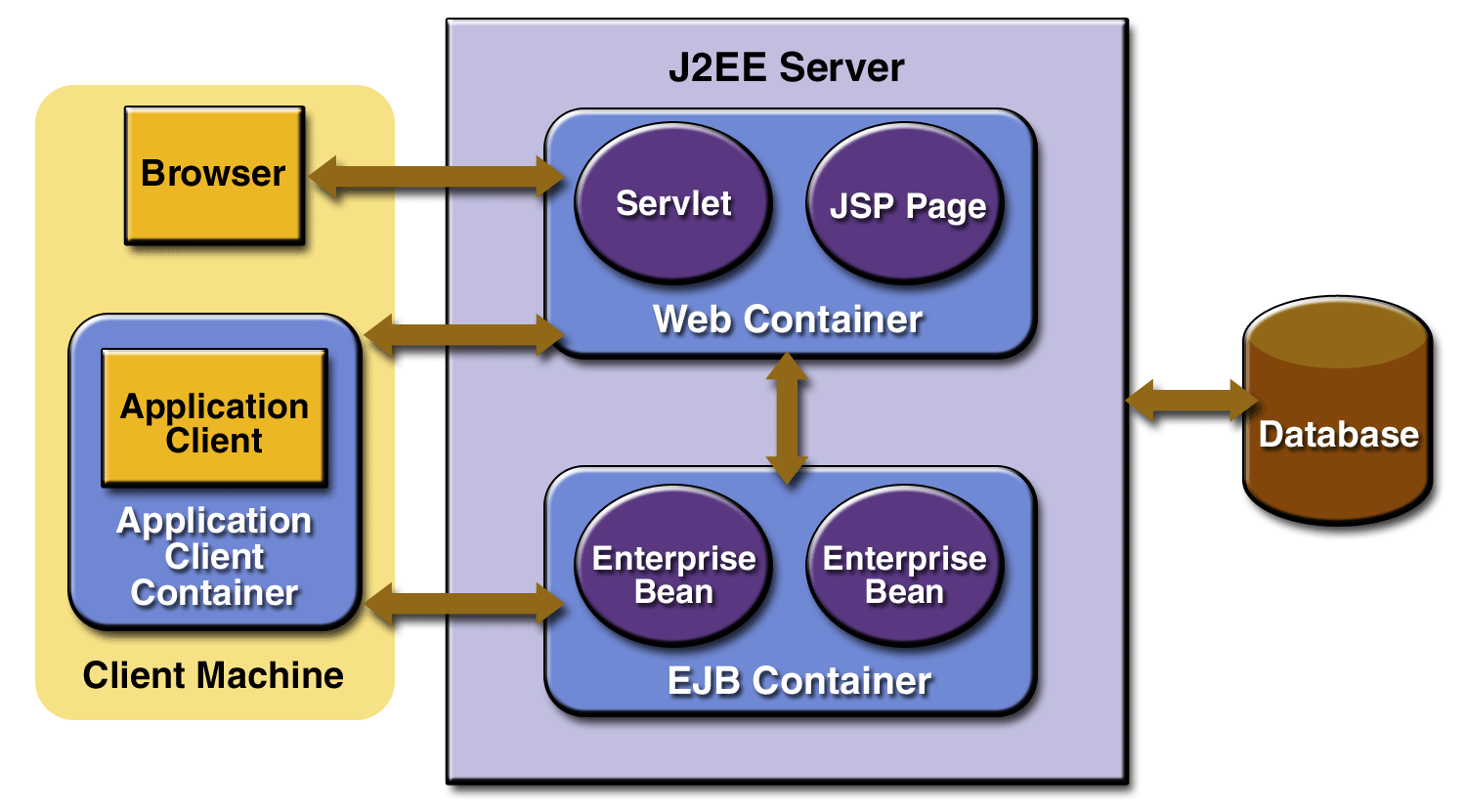
* The J2EE security model lets you configure a Web component or enterprise bean so that system resources are accessed only by authorized users.
* The J2EE transaction model lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.
* JNDI lookup services provide a unified interface to multiple naming and directory services in the enterprise so that application components can access naming and directory services.
* The J2EE remote connectivity model manages low-level communications between clients and enterprise beans. After an enterprise bean is created, a client invokes methods on it as if it were in the same virtual machine.

The fact that the J2EE architecture provides configurable services means that application components within the same J2EE application can behave differently based on where they are deployed. For example, an enterprise bean can have security settings that allow it a certain level of access to database data in one production environment and another level of database access in another production environment.

The container also manages nonconfigurable services such as enterprise bean and servlet life cycles, database connection resource pooling, data persistence, and access to the J2EE platform APIs described in the section [J2EE APIs](http://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/Overview6.html#65514). Although data persistence is a nonconfigurable service, the J2EE architecture lets you override container-managed persistence by including the appropriate code in your enterprise bean implementation when you want more control than the default container-managed persistence provides. For example, you might use bean-managed persistence to implement your own finder (search) methods or to create a customized database cache.

**Container Types**

The deployment process installs J2EE application components in the J2EE containers illustrated in [Figure 1-5](http://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/Overview3.html#78872).



**Figure 1-5 J2EE Server and Containers**

**J2EE server**

The runtime portion of a J2EE product. A J2EE server provides EJB and Web containers.

**Enterprise JavaBeans (EJB) container**

Manages the execution of enterprise beans for J2EE applications. Enterprise beans and their container run on the J2EE server.

**Web container**

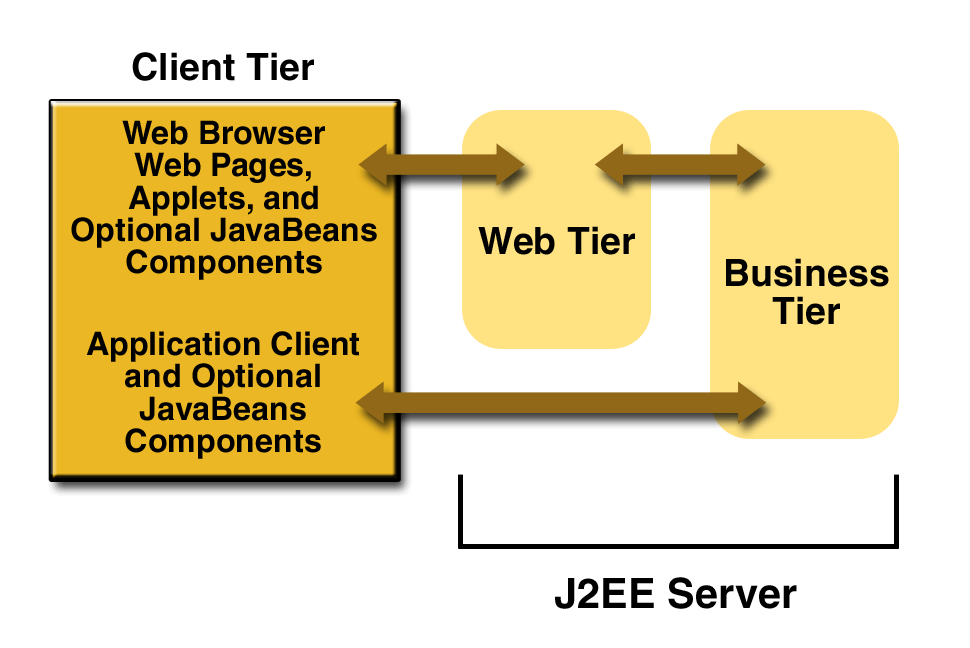
Manages the execution of JSP page and servlet components for J2EE applications. Web components and their container run on the J2EE server.

**Application client container**

Manages the execution of application client components. Application clients and their container run on the client.

**Applet container**

Manages the execution of applets. Consists of a Web browser and Java Plug-in running on the client together.



**Figure 1-2 Server Communications**

**Web Components**

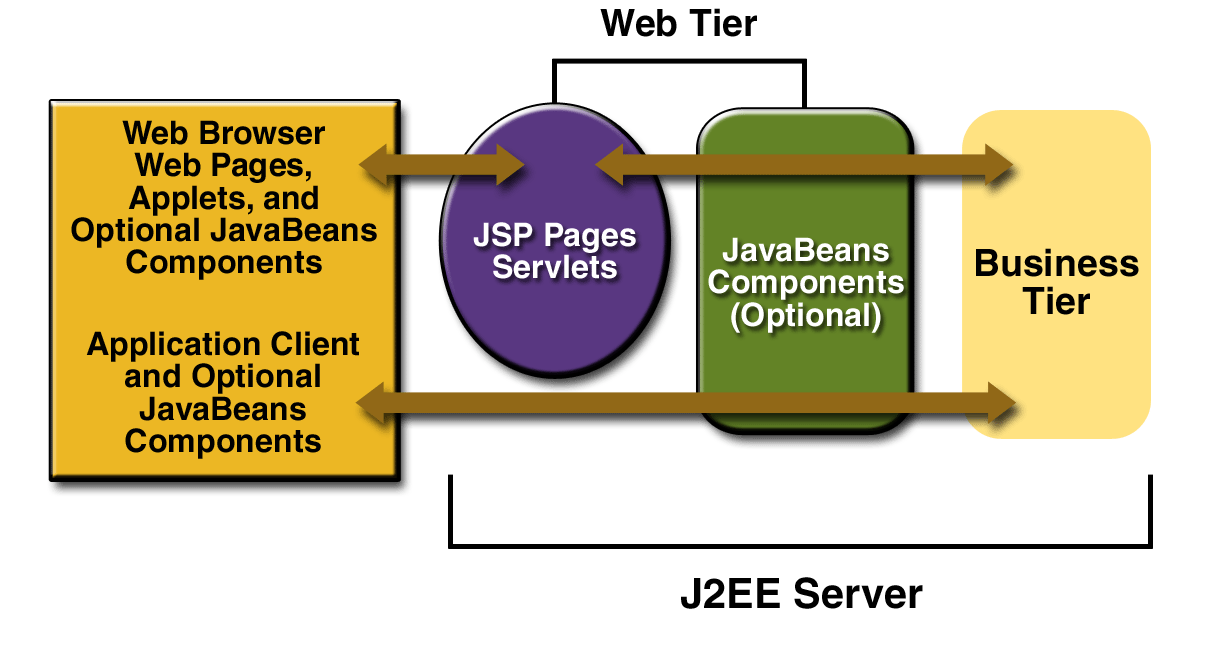
J2EE Web components can be either servlets or JSP pages. *Servlets* are Java programming language classes that dynamically process requests and construct responses. *JSP pages* are text-based documents that execute as servlets but allow a more natural approach to creating static content.

Static HTML pages and applets are bundled with Web components during application assembly, but are not considered Web components by the J2EE specification. Server-side utility classes can also be bundled with Web components and, like HTML pages, are not considered Web components.

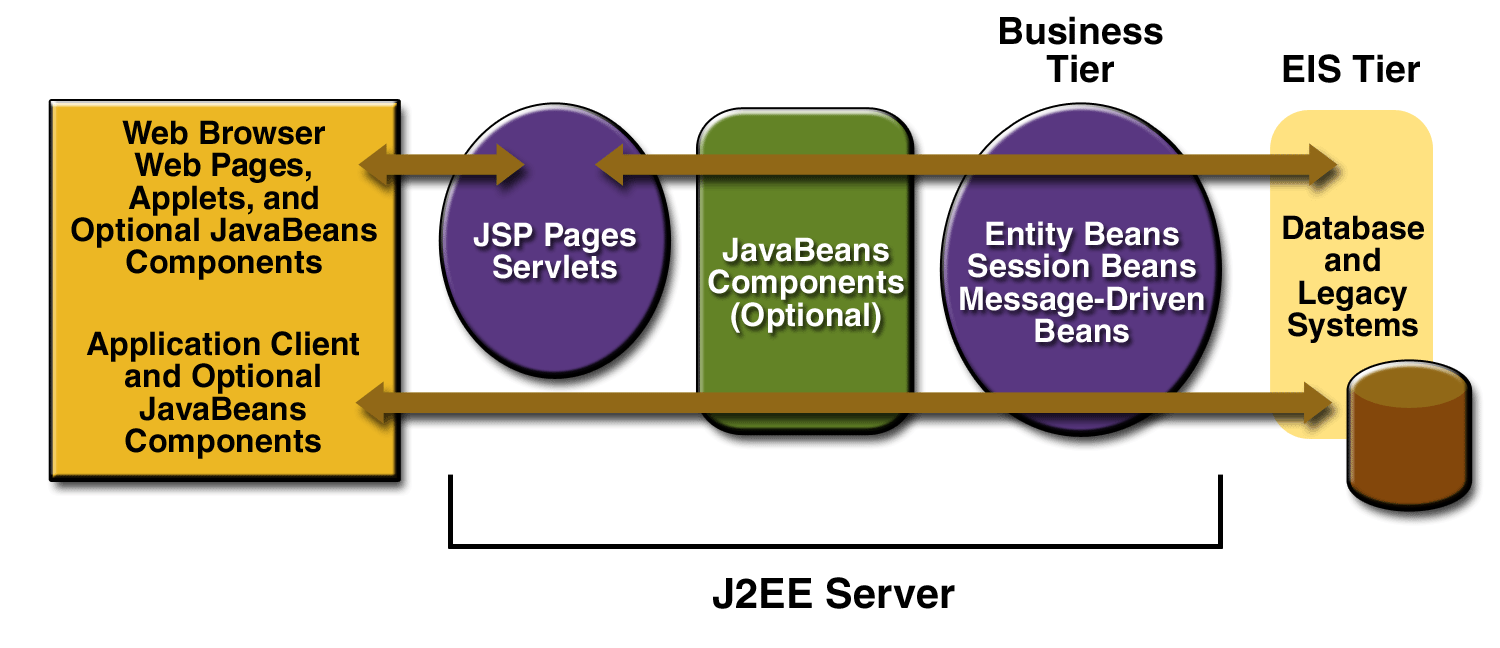
Like the client tier and as shown in [Figure 1-3](http://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/Overview2.html#78856), the Web tier might include a JavaBeans component to manage the user input and send that input to enterprise beans running in the business tier for processing.

**Business Components**

Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in the business tier. [Figure 1-4](http://www.muskingum.edu/~reichard/J2EE/j2eetutorial/doc/Overview2.html#72422) shows how an enterprise bean receives data from client programs, processes it (if necessary), and sends it to the enterprise information system tier for storage. An enterprise bean also retrieves data from storage, processes it (if necessary), and sends it back to the client program.



**Figure 1-3 Web Tier and J2EE Application**



**Figure 1-4 Business and EIS Tiers**

There are three kinds of enterprise beans: session beans, entity beans, and message-driven beans. A *session bean* represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone. In contrast, an *entity bean* represents persistent data stored in one row of a database table. If the client terminates or if the server shuts down, the underlying services ensure that the entity bean data is saved.

A *message-driven bean* combines features of a session bean and a Java Message Service ("JMS") message listener, allowing a business component to receive JMS messages asynchronously. This tutorial describes entity beans and session beans. For information on message-driven beans, see *The Java Message Service Tutorial*, available at

[http://java.sun.com/products/jms/tutorial/index.html](http://java.sun.com/products/jms/tutorial/index.html" \t "_blank)

**Enterprise Information System Tier**

The enterprise information system tier handles enterprise information system software and includes enterprise infrastructure systems such as enterprise resource planning (ERP), mainframe transaction processing, database systems, and other legacy information systems. J2EE application components might need access to enterprise information systems for database connectivity, for example.