

# Java Platform, Enterprise Edition (Java EE)

## 1 Overview

This chapter introduces you to Java EE enterprise application development. Here you will review development basics, learn about the Java EE architecture and APIs, become acquainted with important terms and concepts, and find out how to approach Java EE application programming, assembly, and deployment.

Developers today increasingly recognize the need for distributed, transactional, and portable applications that leverage the speed, security, and reliability of server-side technology. **Enterprise applications** provide the business logic for an enterprise. They are centrally managed and often interact with other enterprise software. In the world of information technology, enterprise applications must be designed, built, and produced for less money, with greater speed, and with fewer resources.

With the Java Platform, Enterprise Edition (Java EE), development of Java enterprise applications has never been easier or faster. The aim of the Java EE platform is to provide developers with a powerful set of APIs while shortening development time, reducing application complexity, and improving application performance.

The Java EE platform is developed through the Java Community Process (JCP), which is responsible for all Java technologies. Expert groups composed of interested parties have created Java Specification Requests (JSRs) to define the various Java EE technologies. The work of the Java Community under the JCP program helps to ensure Java technology's standards of stability and cross-platform compatibility.

The Java EE platform uses a simplified programming model. XML deployment descriptors are optional. Instead, a developer can simply enter the information as an **annotation** directly into a Java source file, and the Java EE server will configure the component at deployment and runtime. These annotations are generally used to embed in a program data that would otherwise be furnished in a deployment descriptor. With annotations, you put the specification information in your code next to the program element affected.

In the Java EE platform, dependency injection can be applied to all resources a component needs, effectively hiding the creation and lookup of resources from application code. Dependency injection can be used in Enterprise JavaBeans (EJB) containers, web containers, and application clients. Dependency injection allows the Java EE container to automatically insert references to other required components or resources, using annotations.

This tutorial uses examples to describe the features available in the Java EE platform for developing enterprise applications. Whether you are a new or experienced enterprise developer, you should find the examples and accompanying text a valuable and accessible knowledge base for creating your own solutions.

The following topics are addressed here:

- [Java EE 7 Platform Highlights](#)
- [Java EE Application Model](#)
- [Distributed Multitiered Applications](#)

- [Java EE Containers](#)
- [Web Services Support](#)
- [Java EE Application Assembly and Deployment](#)
- [Java EE 7 APIs](#)
- [Java EE 7 APIs in the Java Platform, Standard Edition 7](#)
- [GlassFish Server Tools](#)

## 1.1 Java EE 7 Platform Highlights

The most important goal of the Java EE 7 platform is to simplify development by providing a common foundation for the various kinds of components in the Java EE platform. Developers benefit from productivity improvements with more annotations and less XML configuration, more Plain Old Java Objects (POJOs), and simplified packaging. The Java EE 7 platform includes the following new features:

- New technologies, including the following:
  - [Batch Applications for the Java Platform](#)
  - [Concurrency Utilities for Java EE](#)
  - [Java API for JSON Processing](#) (JSON-P)
  - [Java API for WebSocket](#)
- New features for EJB components (see [Enterprise JavaBeans Technology](#) for details)
- New features for servlets (see [Java Servlet Technology](#) for details)
- New features for JavaServer Faces components (see [JavaServer Faces Technology](#) for details)
- New features for the Java Message Service (JMS) (see [Java Message Service API](#) for details)

## 1.2 Java EE Application Model

The Java EE application model begins with the Java programming language and the Java virtual machine. The proven portability, security, and developer productivity they provide form the basis of the application model. Java EE is designed to support applications that implement enterprise services for customers, employees, suppliers, partners, and others who make demands on or contributions to the enterprise. Such applications are inherently complex, potentially accessing data from a variety of sources and distributing applications to a variety of clients.

To better control and manage these applications, the business functions to support these various users are conducted in the middle tier. The middle tier represents an environment that is closely controlled by an

enterprise's information technology department. The middle tier is typically run on dedicated server hardware and has access to the full services of the enterprise.

The Java EE application model defines an architecture for implementing services as multitier applications that deliver the scalability, accessibility, and manageability needed by enterprise-level applications. This model partitions the work needed to implement a multitier service into the following parts:

- The business and presentation logic to be implemented by the developer
- The standard system services provided by the Java EE platform

The developer can rely on the platform to provide solutions for the hard systems-level problems of developing a multitier service.

## Distributed Multitiered Applications

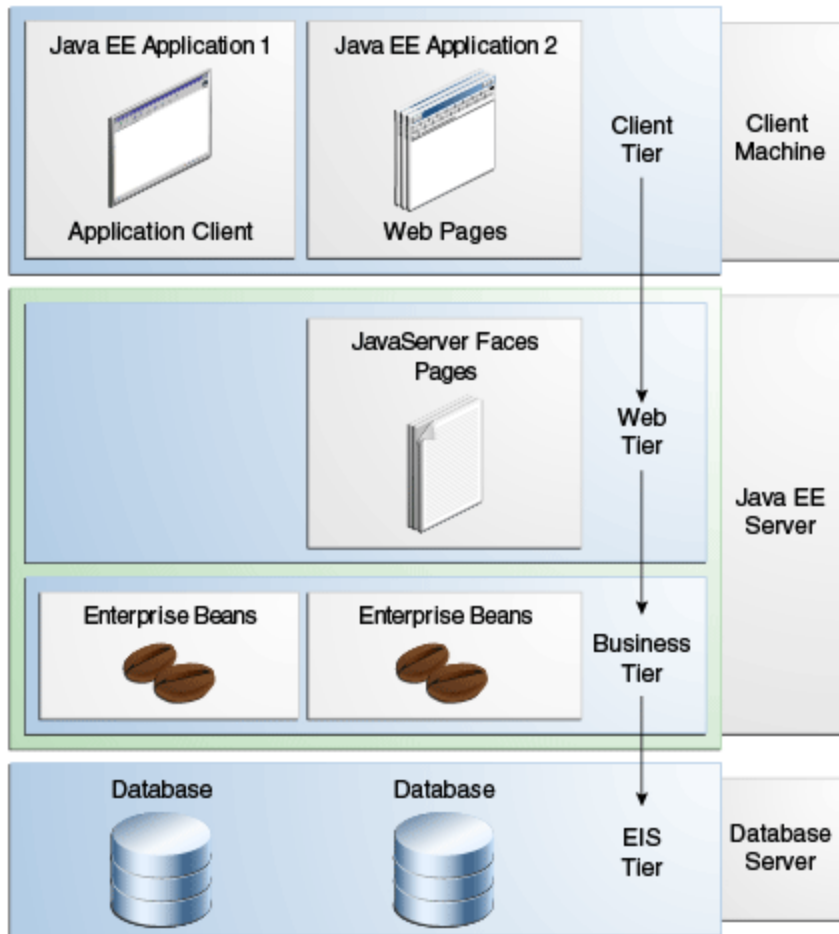
The Java EE platform uses a distributed multitiered application model for enterprise applications. Application logic is divided into components according to function, and the application components that make up a Java EE application are installed on various machines depending on the tier in the multitiered Java EE environment to which the application component belongs.

[Figure 1-1](#) shows two multitiered Java EE applications divided into the tiers described in the following list. The Java EE application parts shown in [Figure 1-1](#) are presented in [Java EE Components](#).

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

Although a Java EE application can consist of all tiers shown in [Figure 1-1](#), Java EE multitiered applications are generally considered to be three-tiered applications because they are distributed over three locations: client machines, the Java EE 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***



[Description of "Figure 1-1 Multitiered Applications"](#)

### 1.3.1 Security

Although other enterprise application models require platform-specific security measures in each application, the Java EE security environment enables security constraints to be defined at deployment time. The Java EE platform makes applications portable to a wide variety of security implementations by shielding application developers from the complexity of implementing security features.

The Java EE platform provides standard declarative access control rules that are defined by the developer and interpreted when the application is deployed on the server. Java EE also provides standard login mechanisms so that application developers do not have to implement these mechanisms in their applications. The same application works in a variety of security environments without changing the source code.

### 1.3.2 Java EE Components

Java EE applications are made up of components. A **Java EE component** is a self-contained functional software unit that is assembled into a Java EE application with its related classes and files and that communicates with other components.

The Java EE specification defines the following Java EE components:

- Application clients and applets are components that run on the client.

- Java Servlet, JavaServer Faces, and JavaServer Pages (JSP) technology components are web components that run on the server.
- EJB components (enterprise beans) are business components that run on the server.

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

### 1.3.3 Java EE Clients

A Java EE client is usually either a web client or an application client.

#### 1.3.3.1 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
- 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 query databases, execute complex business rules, or connect to legacy applications. When you use a thin client, such heavyweight operations are off-loaded to enterprise beans executing on the Java EE server, where they can leverage the security, speed, services, and reliability of Java EE server-side technologies.

#### 1.3.3.2 Application Clients

An **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. An application client typically has a graphical user interface (GUI) created from the Swing API or the Abstract Window Toolkit (AWT) API, 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, an application client can open an HTTP connection to establish communication with a servlet running in the web tier. Application clients written in languages other than Java can interact with Java EE servers, enabling the Java EE platform to interoperate with legacy systems, clients, and non-Java languages.

#### 1.3.3.3 Applets

A web page received from the web tier can include an embedded applet. Written in the Java programming language, an **applet** is a small client application 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 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.

### 1.3.3.4 The JavaBeans Component Architecture

The server and client tiers might also include components based on the JavaBeans component architecture (JavaBeans components) to manage the data flow between the following:

- An application client or applet and components running on the Java EE server
- Server components and a database

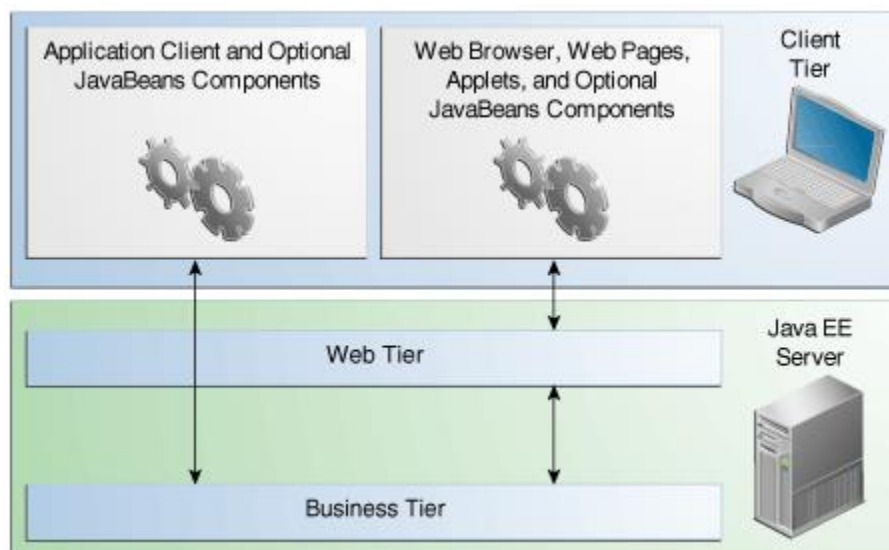
JavaBeans components are not considered Java EE components by the Java EE specification.

JavaBeans components have properties and have `get` and `set` methods for accessing those properties. 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.

### 1.3.3.5 Java EE Server Communications

[Figure 1-2](#) shows the various elements that can make up the client tier. The client communicates with the business tier running on the Java EE server either directly or, as in the case of a client running in a browser, by going through web pages or servlets running in the web tier.

**Figure 1-2 Server Communication**



[Description of "Figure 1-2 Server Communication"](#)

### 1.3.4 Web Components

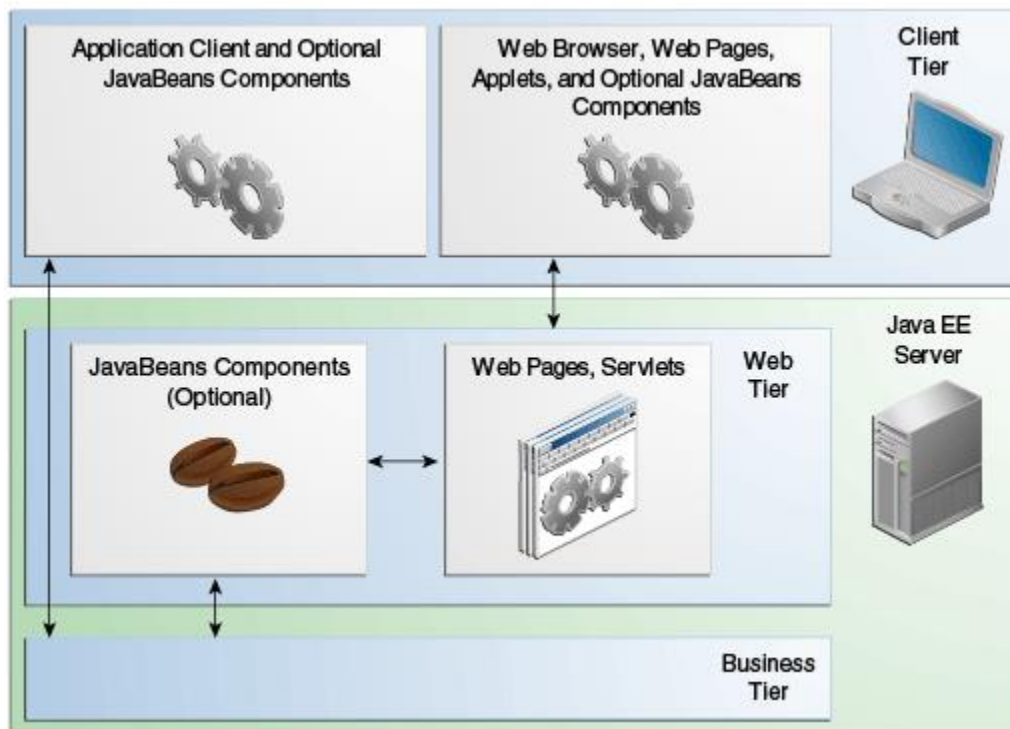
Java EE web components are either servlets or web pages created using JavaServer Faces technology and/or JSP technology (JSP pages). **Servlets** are Java programming language classes that dynamically process requests and construct responses. **JSP pages** are text-based documents that execute as servlets but allow a more natural approach to creating static content. **JavaServer Faces**

**technology** builds on servlets and JSP technology and provides a user interface component framework for web applications.

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

As shown in [Figure 1-3](#), the web tier, like the client 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.

**Figure 1-3 Web Tier and Java EE Applications**

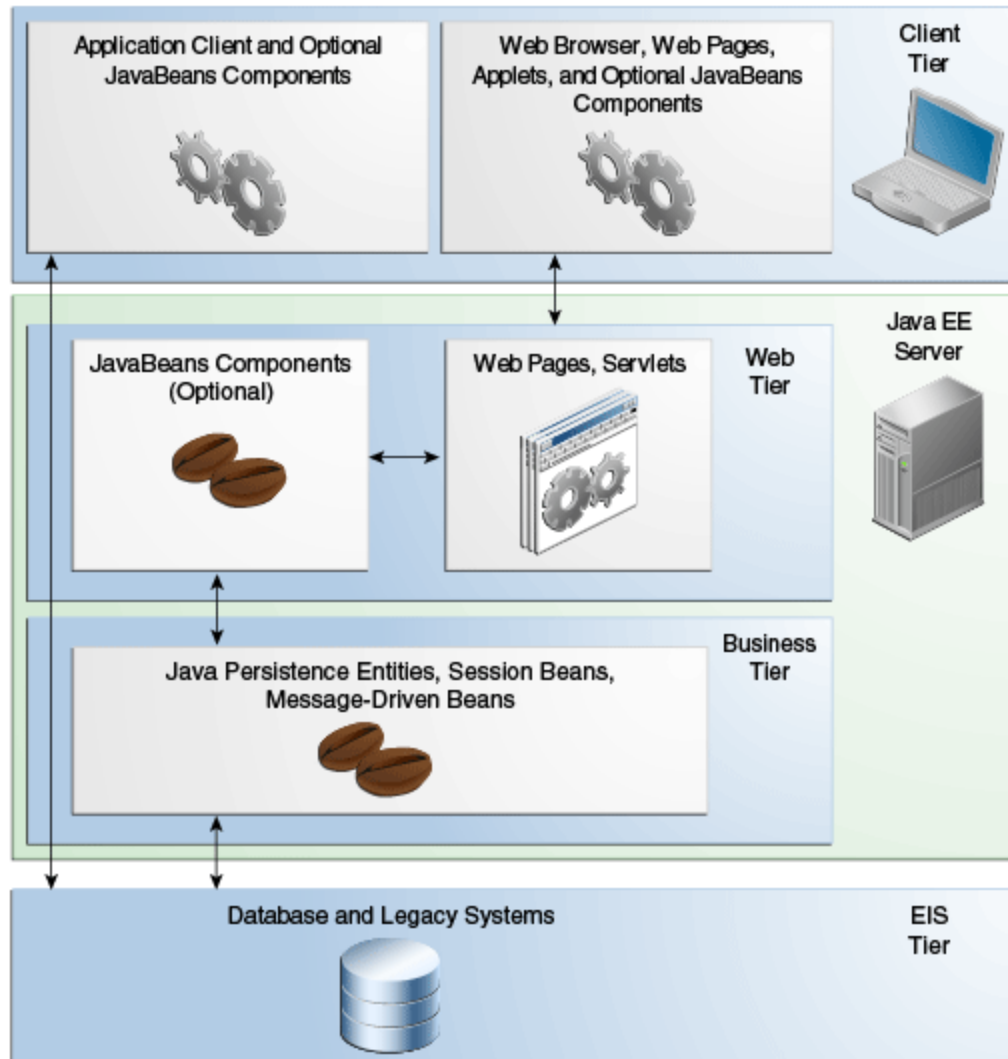


[Description of "Figure 1-3 Web Tier and Java EE Applications"](#)

### 1.3.5 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 either the business tier or the web tier. [Figure 1-4](#) 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 1-4 Business and EIS Tiers**



[Description of "Figure 1-4 Business and EIS Tiers"](#)

### 1.3.6 Enterprise Information System Tier

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

## 1.9 GlassFish Server Tools

GlassFish Server is a compliant implementation of the Java EE 7 platform. In addition to supporting all the APIs described in the previous sections, GlassFish Server includes a number of Java EE tools that are not part of the Java EE 7 platform but are provided as a convenience to the developer.

This section briefly summarizes the tools that make up GlassFish Server. Instructions for starting and stopping GlassFish Server, starting the Administration Console, and starting and stopping the Java DB server are in [Chapter 2, "Using the Tutorial Examples"](#).



GlassFish Server contains the tools listed in [Table 1-1](#). Basic usage information for many of the tools appears throughout the tutorial. For detailed information, see the online help in the GUI tools.

**Table 1-1 GlassFish Server Tools**

Tool	Description
Administration Console	A web-based GUI GlassFish Server administration utility. Used to stop GlassFish Server and to manage users, resources, and applications.
<code>asadmin</code>	A command-line GlassFish Server administration utility. Used to start and stop GlassFish Server and to manage users, resources, and applications.
<code>appclient</code>	A command-line tool that launches the application client container and invokes the client application packaged in the application client JAR file.
<code>capture-schema</code>	A command-line tool to extract schema information from a database, producing a schema file that GlassFish Server can use for container-managed persistence.
<code>package-appclient</code>	A command-line tool to package the application client container libraries and JAR files.
Java DB database	A copy of the Java DB server.
<code>xjc</code>	A command-line tool to transform, or bind, a source XML schema to a set of JAXB content classes in the Java programming language.
<code>schemagen</code>	A command-line tool to create a schema file for each namespace referenced in your Java classes.
<code>wsimport</code>	A command-line tool to generate JAX-WS portable artifacts for a given WSDL file. After generation, these artifacts can be packaged in a WAR file with the WSDL and schema documents, along with the endpoint implementation, and then deployed.
<code>wsgen</code>	A command-line tool to read a web service endpoint class and generate all the required JAX-WS portable artifacts for web service deployment and invocation.