Java Enterprise Edition (JEE)

JEE is a collection of many different specifications intended to perform specific tasks. These specifications are defined by the Java Community Process (https://www.jcp.org (https://www.jcp.org)) program. Currently, JEE is in version 7. However, different specifications of JEE are at their own different versions.

JEE specifications can be broadly classified in the following groups:

- Presentation layer
- Business layer
- Enterprise integration layer

Note that JEE specification does not necessarily classify APIs in such broad groups, but such classification could help in better understanding the purpose of the different standards and APIs in JEE. Before we see APIs in each of these categories, let's understand a typical JEE web application flow where each of these layers fits in.

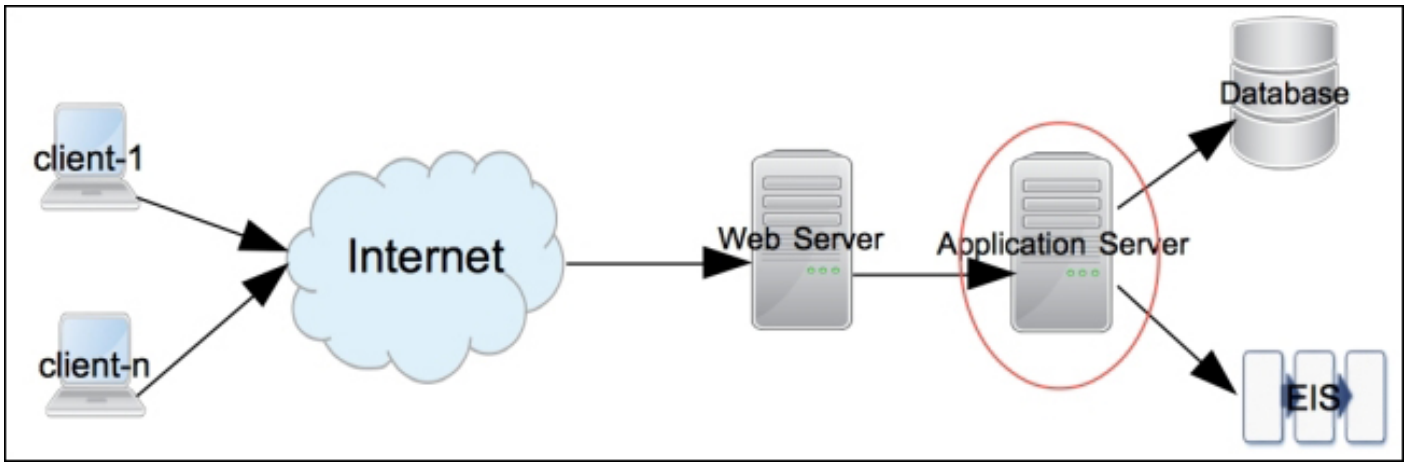

Figure 1.1 A typical JEE web application flow

Requests start from client. Client can be any application requesting services from a remote application – for example, it could be a browser or a desktop application. The request is first received by Web Server at the destination. Examples of Web Servers are Apache Web Server, IIS, Nginx, and so on. If it is a request for static content, then it is served by the web server(s). However, dynamic request typically requires an Application Server to process it. JEE servers are such Application Servers that handle the dynamic requests. Most JEE specification APIs execute in the application server. Examples of JEE application servers are WebLogic, WebSphere, GlassFish, JBoss, and so on.

Most non-trivial JEE applications access external systems such as database or **Enterprise Integration Server** (**EIS**) for data and process it.

Response is returned from the application server to the web server and then to the clients.

The following is the brief description of each of the JEE specifications in different layers of applications that we saw previously. We will see how to use these APIs in more detail in subsequent chapters. However, note that the following is not the exhaustive list of all the specifications in JEE. We will see the most commonly used specifications here. For the exhaustive list, please visit http://www.oracle.com/technetwork/java/javaee/tech/index.html (http://www.oracle.com/technetwork/java/javaee/tech/index.html).

The presentation layer

JEE specifications or technologies in this group receive the request from web server and send back the response, typically, in an HTML format. However, it is also possible to return only the data from the presentation layer, for example, in **JavaScript Object Notation** (**JSON**) or **eXtensible Markup Language** (**XML**) format, which could be consumed by AJAX (Asynchronous JavaScript and XML) calls to update only part of the page, instead of rendering the entire HTML page. Classes in the presentation layer are mostly executed in a Web Container – it is a part of the application server that handles web requests. Tomcat is an example of a popular Web Container.

Now, we will take a look at some of the specifications in this group.

Java Servlet

Java servlets are server side modules, typically used to process a request and send back response in the web applications. Servlets are useful for handling requests that do not generate large HTML markup responses. They are typically used as controllers in MVC (Model View Controller) frameworks, for forwarding/redirecting requests or for generating non-HTML responses, such as PDFs. To generate an HTML response from Servlet, you need to embed the HTML code (as Java String) in the Java code. Therefore, it is not the most convenient option for generating large HTML response. JEE 7 contains Servlet API 3.1.

Java Server Pages

Like Servlets, JSPs are also server side modules used to process the web requests. JSPs (Java Server Pages) are great for handling requests that generate large HTML markup responses. In JSP pages, Java code or JSP tags can be mixed with other HTML code, such as HTML tags, JavaScript, and CSS. Since Java code is embedded in the larger HTML code, it is easier (than Servlet) to generate an HTML response from the JSP pages. JSP specification 2.3 is included in JEE 7.

Java Server Faces

Java Server Faces makes creating user interface on the server side modular by incorporating the MVC design pattern in its implementation. It also provides easy to use tags for common user interface controls that can save states across multiple request-response exchanges between the client and server. For example, if you have a page that posts form data from a browser, you can have JSF save that data in a Java Bean so that it can be

used subsequently in the response to the same or different request. JSF also makes it easier to handle UI events on the server side and specify page navigation in an application.

You write the **Java Server Faces** (**JSF**) code in JSP, using custom JSP tags created for JSF. Java Server Faces API 2.2 is part of JEE 7.

The business layer

The business layer is where you typically write code to handle the business logic of your application. The request to this layer could come from the presentation layer, directly from the client application, or from the middle layer consisting of, but not limited to, web services. Classes in this layer are executed in the application container part of JEE Server. GlassFish and WebSphere are examples of web container plus application container.

Let us take a tour of some of the specifications in this group.

Enterprise Java Beans

Enterprise Java Beans (**EJBs**) are the Java classes where you can write your business logic. Though it is not a strict requirement to use EJBs to write business logic, they do provide many of the services that are essential in enterprise applications. These services are security, transaction management, component lookup, object pooling, and so on. You can have EJBs distributed across multiple servers and let the application container (also called EJB container) take care of component look up (searching component) and component pooling (useful for scalability). This can improve scalability of the application.

EJBs are of two types:

- **Session beans**: Session beans are called directly by clients or middle tier objects
- **Message driven beans**: Message driven beans are called in response to **Java Messaging Service** (**JSM**) events

JMS and message driven beans can be used for handling asynchronous requests. In a typical asynchronous request processing scenario, the client puts a request in a messaging queue or a topic and does not wait for immediate response. Server application gets the request message, either directly using JMS APIs or by using MDB. It processes the request and may put response in a different queue or topic to which the client would listen and get the response.

Java EE 7 contains EJB specification 3.2 and JMS specification 2.0.

The enterprise integration layer

APIs in this layer are used for interacting with external (to JEE application) systems in Enterprise. Most applications would need to access database, and APIs to access it fall in this group.

Java Database Connectivity (JDBC)

JDBC is a specification to access relational database in a common and consistent way. Using JDBC you can execute SQL statements and get results on different databases using common APIs. Database specific driver sits between the JDBC call and the database, which translates JDBC calls to database vendor specific API calls. JDBC can be used in both the Presentation and Business layers directly, but it is recommended to separate the database calls from both UI and the business code. Typically, this is done by creating **Data Access Objects** (**DAO**) which encapsulate logic to access the database.

JEE 7 contains JDBC specification 4.0.

The Java Persistent API (JPA)

One of the problems of using JDBC APIs directly is that you have to constantly map the data between Java Objects and the data in columns of rows in relational database. Frameworks such as Hibernate and Spring have made this process simpler by using a concept known as **Object Relationship Mapping** (**ORM**). ORM is incorporated in JEE in the form of **Java Persistent API** (**JPA**). JPA gives you the flexibility to map the objects to the tables in relational database and execute the queries with or without using **Structured Query Language** (**SQL**). Though when used in the content of JPA, query language is called Java Persistence Query Language. JPA specification 2.1 is a part of JEE.

Java Connector Architecture (JCA)

JCA APIs can be used in JEE applications for communicating with Enterprise Integration Systems, such as SAP, Salesforce, and so on. Just like you have database drivers to broker communication between JDBC APIs and relational database, you have JCA adapters between JCA calls and EIS. Most EIS applications now provide REST APIs, which are lightweight and easy to use, so REST could replace JCA in some cases. However, if you use JCA, you get transaction and pooling support from JEE application server.

Web services

Web services are remote application components that expose self-contained APIs. Broadly, web services can be classified based on the following two standards:

- **Simple Object Access Protocol** (**SOAP**)
- **Representational State Transfer** (**REST**)

Web services can play a major role in integrating disparate applications, because they are standard based and platform independent.

JEE provides many specifications to simplify development and consumption of both types of web services, for example, JAX-WS (Java API for XML – web services) and JAX-RS (Java API for RESTful web services).

The preceding are just some of the specifications that are part of JEE. There are many other independent specifications, such as web services, and many enabling specifications, such as dependency injection and

concurrency utilities, that we will see in subsequent chapters.

Eclipse IDE

As mentioned earlier, a good IDE is essential for better productivity while coding. Eclipse is one such IDE, which has great editor features and many integration points with JEE technologies. The primary purpose of this book is to show you how to develop JEE applications using Eclipse. So following is a quick introduction to Eclipse, if you are not already familiar with it.

Eclipse is an open source IDE for developing applications in many different programming languages. It is quite popular for developing many different types of Java applications. Its architecture is pluggable – there is a core IDE and many different plugins can be added to it. In fact, support for many languages is added as Eclipse plugins, including support for Java.

Along with editor support, Eclipse has plugins to interact with many of the external systems used during development. For example, source control systems such as SVN and Git, build tools such as Apache Ant and Maven, file explorer for remote systems using FTP, managing servers such as Tomcat and GlassFish, database explorer, memory and CPU profiler, and so on. We will see many of these features in the subsequent chapters.

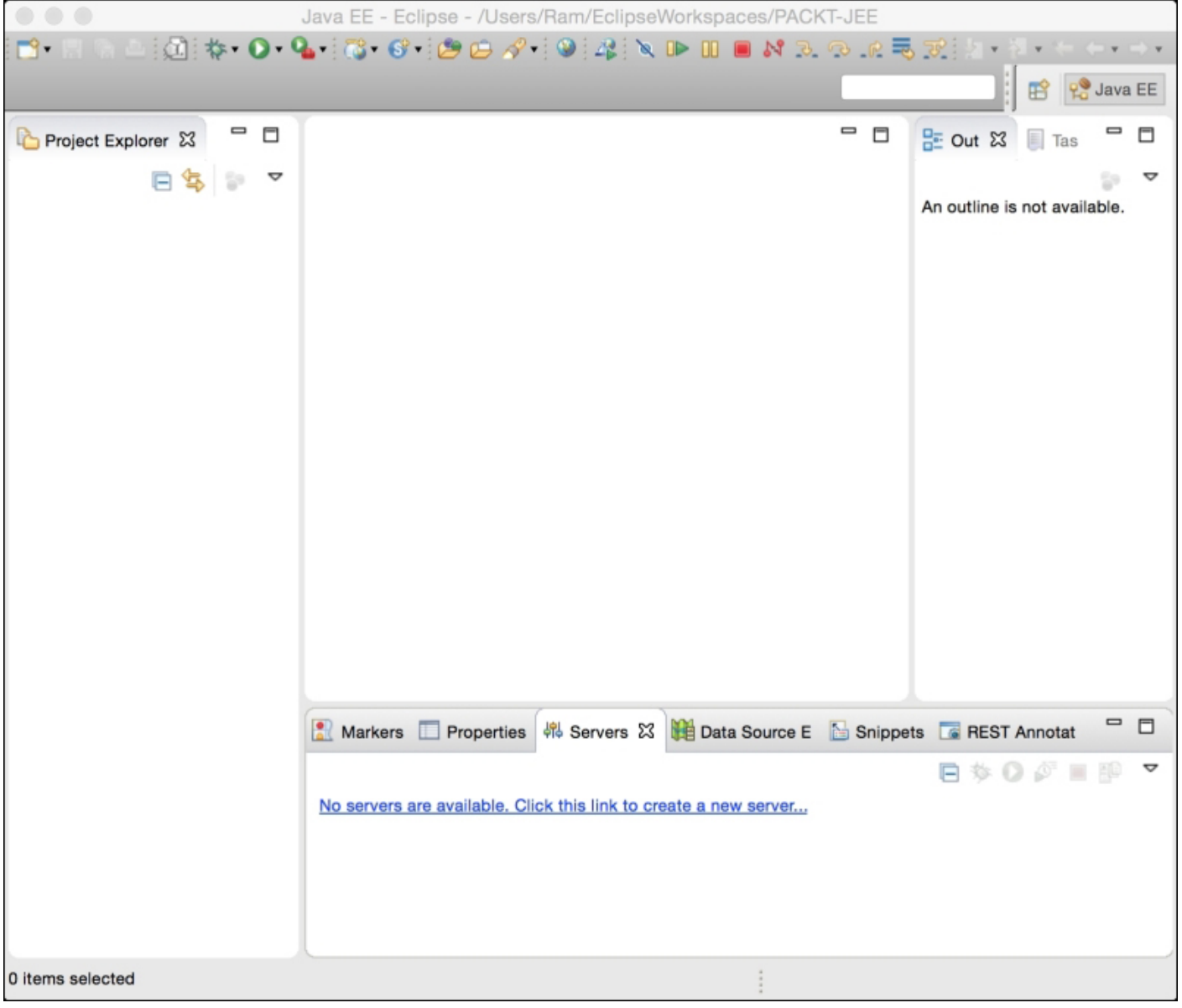

Figure 1.2 Default Eclipse View

Figure 1.2 shows the default view of Eclipse for JEE application development. When working with Eclipse, it is good to understand the following terms used in the context of Eclipse.

Workspace

The Eclipse workspace is a collection of projects, settings, and preferences. Workspace is a folder where Eclipse stores this information. You must create a workspace to use Eclipse. You can create multiple workspaces, but

at a time only one can be opened by one running instance of Eclipse. However, you can launch multiple instances of Eclipse with different workspaces.

Plugin

Eclipse has pluggable architecture. Many of the features of Eclipse are implemented as plugins, for example, editor plugins for Java and many other languages, plugins for SVN and Git, and many others. Default installation of Eclipse comes with many built-in plugins and you can add more plugins for the features you want later.

Editors and views

Most windows in Eclipse can be classified either as editor or views. Editor is something where you can change the information displayed in it. View just displays the information and does not allow you to change it. An example of an editor is the Java editor where you write a code. An example of view is the outline view that displays the hierarchical structure of the code you are editing (in case of Java editor, it shows classes in a file, and methods in them).

To see all views in a given Eclipse installation, open the **Window** | **Show View** | **Other** menu.

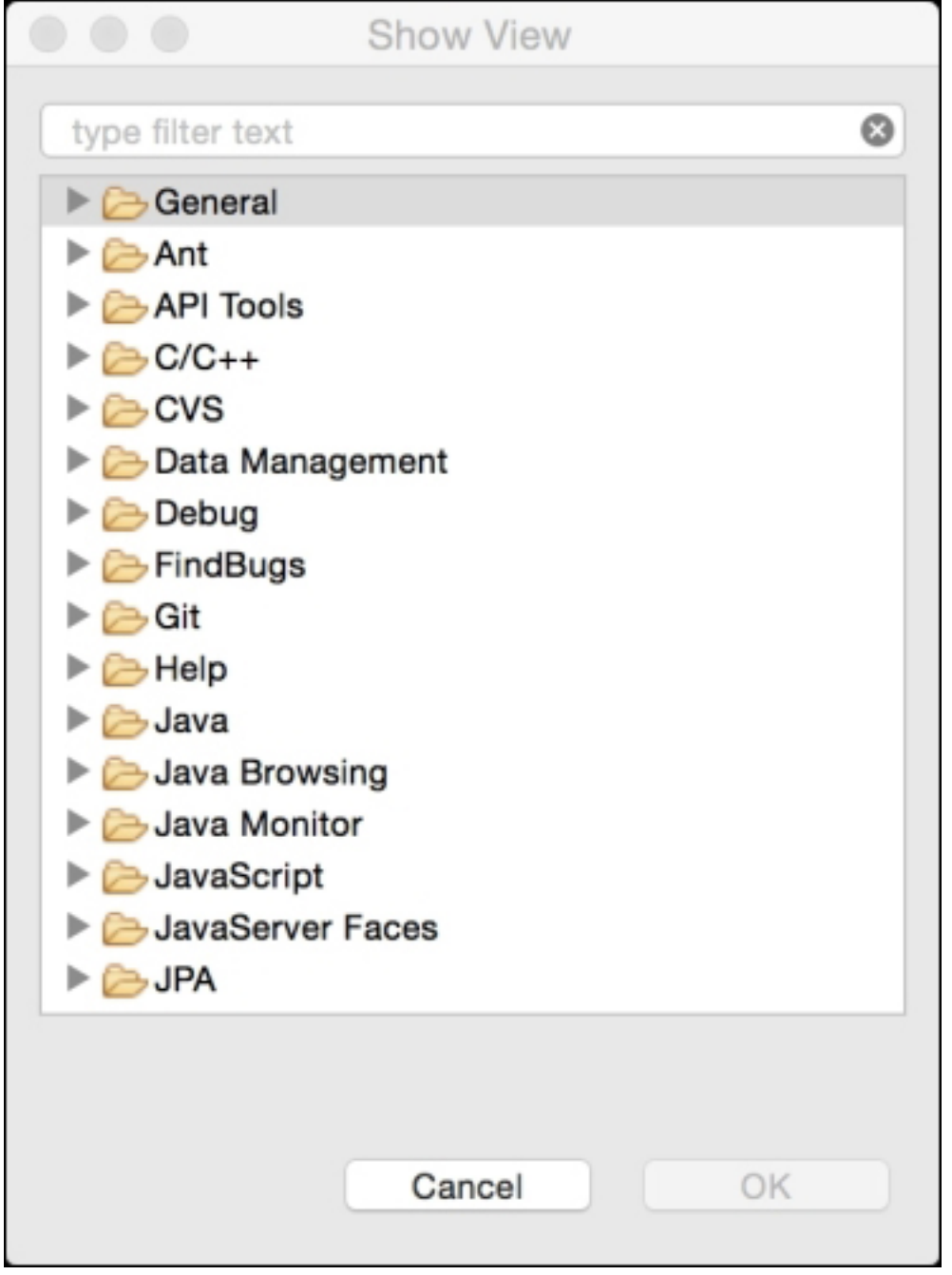

Figure 1.3 Show all Eclipse Views

Perspective

Perspective is a collection of editors and views, and how they are laid out or arranged in the main Eclipse window. At different stages of development, you need different views to be displayed. For example, when you are editing a code, you need to see the **Project Explorer** and **Task** views, but when you are debugging an application, you don't need those

views, but instead want to see the variables and breakpoints view. So, the editing perspective displays, among other views and editor, the **Project Explorer** and **Task** view and the Debug perspective displays views and editors relevant to the debugging activities. You can change the default perspectives to suit your purpose, though.

Eclipse preferences

The Eclipse preferences window is where you customize many features of plugins/features. Preferences are available from the **Window** menu in Windows and Linux installation of Eclipse, and from Eclipse menu in Mac installation of Eclipse.

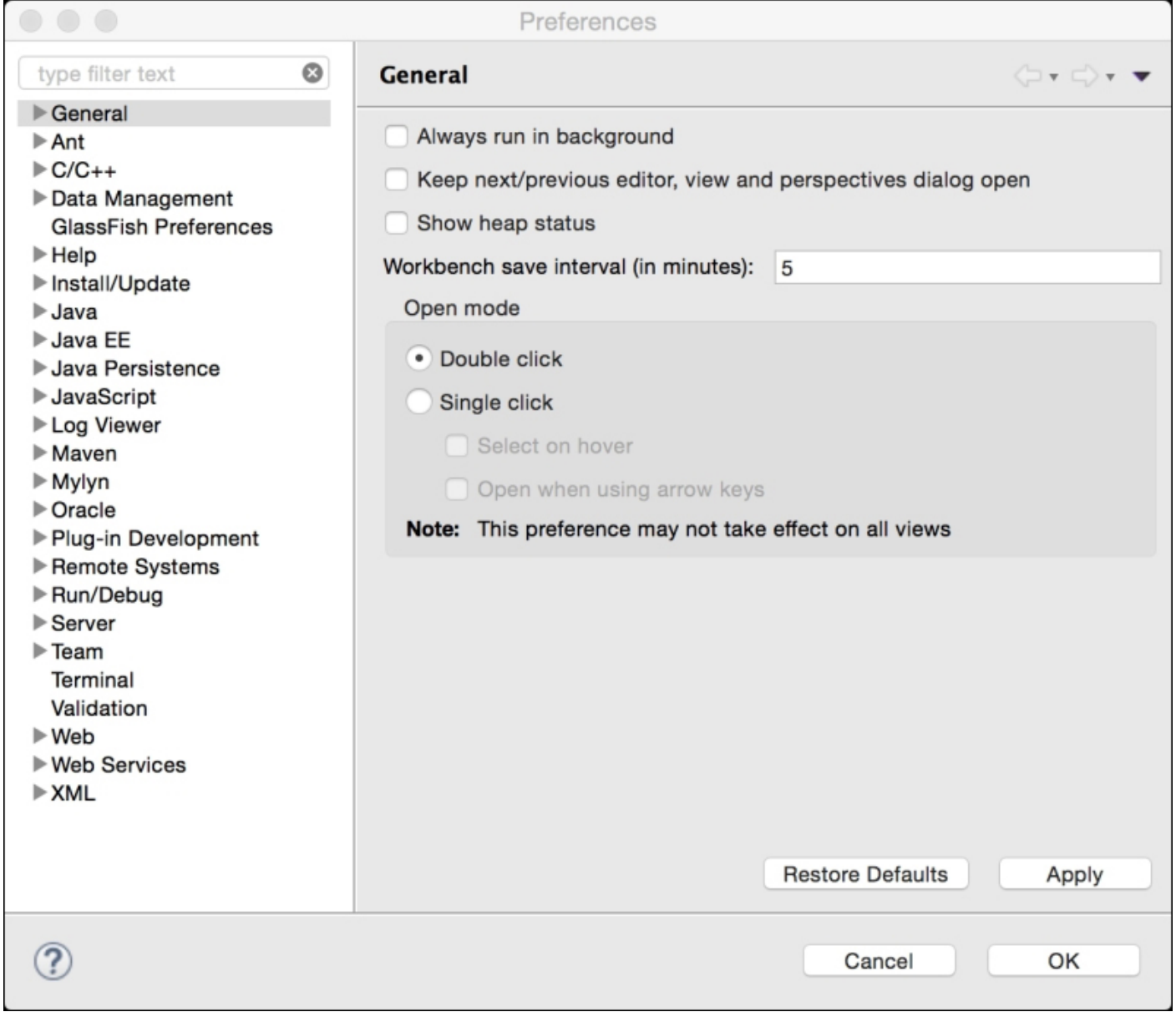

Figure 1.4 Eclipse Preferences

‹ Previous Section (/book/application_development/9781785285349/1

Next Section › (/book/application_development/9781785285349/1/ch