Tomitribe

Apache TomEE Training

COURSE SYLLABUS

Version 1.0.5, 2017-02-12

Overview

The goal of this course is to help senior and mid-level Java developers grasp the concepts, contracts, and technologies in Java EE 7 in order to empower them to write, configure, test, and debug Java EE 7 applications efficiently using the APIs, testing tools, and server runtime provided by Apache TomEE 7. This course will cover such topics as component development (CDI & EJB), RESTful web services (JAX-RS), object-relational mapping (JPA, JTA, & JDBC), messaging (JMS), and in-container testing (Arquillian).

For those coming from Apache Tomcat, the transition should be straightforward since TomEE is just Tomcat with extra Java EE powers. The additional integrations that Java EE 7 and TomEE provide are sure to save developers time, whether it's writing code, testing business logic, configuring container-managed resources, or administering the server. For those with a strong J2EE background, the improvements to the platform are sure to be eye-opening and prove the need for abstractions that hide cumbersome, legacy APIs or patch missing functionality to be obsolete.

Regardless of background or skill level, students will find that this course gives them everything they need to become a modern Java EE 7 developer.

Agenda

Day 1

Morning

Module 1: Java EE 7 Foundations

Java EE 7 offers a modern, integrated, and extensible programming model for developing enterprise applications. Apache TomEE 7 provides a runtime that enables you to start putting these APIs into practice. We'll look at how Java EE 7 displaces the cumbersome J2EE programming model of the past with a lightweight contract based on annotated POJOs, automatic discovery, dependency injection, fluent APIs, and modern architectural principles. You'll learn about the foundation mechanisms that make Java EE 7 tick so you can identify, reason about, and apply the recurring techniques and technologies you'll encounter throughout the course.

Learning objectives

- State the advantages of Java EE 7
- Draw contrast with component models and container services of old
- Be familiar with terminology in Java EE 7
- Define a managed component (aka "bean")

- Understand the mechanism by which components are discovered
- Understand the mechanism by which components are wired together
- Conceptualize how services get added to components
- Recognize the differences between various component types
- Map business use cases to technologies in the platform

Module 2: Apache TomEE Basics

A Java EE application is run on a Java EE application server. Apache TomEE is one such implementation. In the modules that follow, you'll learn Java EE 7 through the lens of TomEE 7. Therefore, you need to be comfortable downloading, extracting, running, and deploying applications to TomEE. You'll also learn how to perform some essential configuration tasks in order to work with TomEE on a daily basis.

Learning objectives

- Understand the relationship between Apache Tomcat and Apache TomEE
- Identify the prerequisites for running TomEE
- Find, select, and download TomEE
- Prepare a standalone server instance
- Prepare a server instance for testing
- Start and stop TomEE in the foreground and the background
- Deploy a web application to TomEE using several approaches
- Enable SSL using a self-signed certificate
- Coarsely tune logging
- Locate the console log when running TomEE in the background

Module 3: Development Tools Orientation

We want you to remain as hands-on as possible throughout this course. After all, a programming model is just theory until you put it to use. You'll use this module to get your local development environment set up. You'll get acclimated to the development workbench we've prepared for you and practice the basic tasks you'll need to carry out the labs. The tools we'll cover include Apache TomEE, Apache Maven, JUnit, Arquillian, and either Eclipse or IntelliJ IDEA.

Learning objectives

- Add the TomEE-provided Java EE APIs to a project
- Build a Java web project using Maven

- Run a Java web application using the TomEE Maven plugin
- Access the running application from the browser
- Import a Maven-based web project into Eclipse or IntelliJ IDEA
- Add a run configuration in the IDE to control TomEE
- Start and stop TomEE from the IDE
- Deploy and redeploy a web application to TomEE from the IDE
- Add Arquillian to a project
- Write a JUnit-based Arquillian test
- Run an Arquillian test on TomEE embedded
- Debug an Arquillian test in the IDE

Afternoon

Module 4: CDI Building Blocks

CDI is the unifying technology in the Java EE 7 programming model. We'll study CDI first so you can master the fundamental concepts, contracts, and abstractions that underpin the Java EE services we'll explore throughout this training.

Learning objectives

- Describe the role of a CDI bean
- List the requirements that make a class eligible to become a CDI bean
- Promote a class to a CDI bean
- Wire CDI beans together using typesafe injection
- Identify valid CDI injection points
- Reason about how a CDI bean gets resolved
- Clarify an ambiguous dependency using a qualifier annotation
- Understand how a CDI bean gets invoked
- Name the default scopes and characterize the lifespans of their corresponding contexts
- Describe the life cycle of a CDI bean and where bean instances are stored
- Access generic resources from the container using resource injection
- Promote an object not managed by CDI to a CDI bean using a producer

Day 2

Morning

Module 5: Stepping up to EJB

As of Java EE 7, EJB still offers critical and convenient services not yet available to CDI beans. This module will teach you about the common and specialized services inherent to EJB session beans (stateless, singleton, stateful). We'll consider when it's appropriate to "upgrade" your CDI bean to access these services. While you can think of these additional services as built-in CDI extensions, there are conditions to be aware of to use them safely and effectively, which we'll cover.

Learning objectives

- Understand where EJBs provide value
- Identify the services uniquely available to EJB
- Identify three types of session beans and their specialties
- Perform the singleton pattern in Java EE
- Invoke code during container initialization
- Control concurrent access to a shared component
- Invoke a method asynchronously
- Understand passivation
- Inject EJBs using @Inject
- Select an appropriate scope per EJB type

Afternoon

Module 6: JAX-RS (RESTful Web Services) Primer

RESTful web services have become the predominant entry point into enterprise applications. They serve as a modern intersection between client and server, allowing native HTTP requests to interact with resources that represent backend data. In this module, you'll learn the basics of creating simple RESTful web services using JAX-RS. These lessons establish a critical foundation for testing and experimenting with managed components.

Learning objectives

- Understand the principles of REST and how they map to JAX-RS
- Enable JAX-RS in a Java EE web application
- Declare and map a RESTful resource
- Understand how a resource method is selected to handle a request path
- Send a response to a RESTful client
- Test a RESTful web service using Arquillian and the JAX-RS client API
- Read parameters and other contextual information from the inbound request

- Map dynamic resource paths
- Partition resource across subresource handlers
- Understand how payload data is converted

Day 3

Morning

Module 7: Getting down to Business

JAX-RS offers an extensive set of resource handlers and providers to adapt data in a backend system to a format the client accepts. Integrating these components to customize and extend a JAX-RS application requires putting into practice all the constructs you've learned thus far, which you'll see working together for the first time. To start, we'll add CDI to the JAX-RS resource and inject components that handle the business logic, then we'll use JAXB to convert between request entities and Java objects. Next, we'll validate inbound and outbound data using Bean Validation. Finally, we'll monitor the information channel to fine-tune responses and handle exceptions when they strike.

Learning objectives

- Appropriately scope a JAX-RS resource
- Use CDI features in a JAX-RS resource
- Negotiate content in a REST request
- Produce XML and JSON responses using JAXB
- Create a custom resource converter
- Register a custom provider
- Validate REST input parameters
- Handle exceptions in a REST request
- Customize the HTTP response code
- Access and customize the WADL for a resource

Afternoon

Module 8: Transacting with the DataSource

At the heart of Java persistence lies a database connection and its transaction. But before you learn how to relate Java objects to database tables, you need to have a grasp of the Java Database Connectivity (JDBC) subsystem. In this module, you'll encounter the javax.sql.DataSource. This container-managed resource holds connection properties, provides a factory for connections to a physical data source (such as a database), and manages the connection pool. You'll learn how to define a new DataSource to connect to a development database. Then, you'll submit queries to study how a set of database operations is automatically governed by a transaction. Finally, you'll deploy a startup component to initialize and seed the database.

Learning objectives

- Access a container-managed DataSource
- Launch HSQLDB or H2 in server mode
- Install an external database driver
- Understand where a DataSource can be defined
- Configure a new DataSource
- Use property replacements in the DataSource configuration
- Characterize the properties of a DataSource in TomEE
- Use a DataSource to work with JDBC
- Save data to a database
- Execute a query and retrieve data from a JDBC ResultSet
- Describe the difference between XA and non-XA transactions and datasources
- Roll back a transaction manually
- Use an exception to roll back a transaction
- Generate a ciphered password to use in the DataSource configuration

Day 4

Morning/Afternoon

Module 9: Relating Objects to the Database

Data is the currency of an enterprise, so you want it flowing through your application in the most seamless way possible. In this module, you'll learn about the suite of faculties the Java Persistence API (JPA) provides to transport data between a database and your application, ranging from object-relational mapping (ORM) to native SQL. You'll learn how to activate JPA, map Java classes to database tables, validate inbound data, and use the persistence context to transform objects into database records and vice versa. We'll then look at the various query mechanisms JPA provides, when each is appropriate, and how to optimize the queries. You'll come to appreciate how JPA carefully blends productivity with scalability.

Learning objectives

- Describe the purpose of JPA and how it relates to data mapping and ORM

- Identify JPA's key capabilities
- Configure and activate JPA
- Map a JPA entity to the database
- Add property-level constraints on a JPA entity
- Synchronize the database schema automatically
- Access the EntityManager
- Persist, fetch and delete a JPA entity
- Understand the behavior of the persistence context
- Update a JPA entity transparently or forcefully
- Choose a locking strategy
- Enable optimistic locking using the version column
- Contrast the various query mechanisms in JPA
- Work with relationships and collections
- Tune lazy loading
- Optimize the fetch plan
- Understand the interplay between the persistence context and transactions

Day 5

Morning/Afternoon

Module 10: Messaging the Enterprise over JMS

Reliable messaging is a powerful tool that can be used to scale an application to a Fortune 500 business. It can provide a second chance to process critical information if a bug appears in your production application. In this module, you'll become familiar with the different styles of asynchronous messaging, how the Java Message Service (JMS) is used to pass messages between managed components (Message-Driven Beans and CDI beans), how to setup and configure JMS in the TomEE and ActiveMQ runtimes, and when to apply this technology to improve the performance, integrity, and availability of the application.

Learning objectives

- Describe messaging types: Point-to-Point (Queues) and Publish/Subscribe (Topics)
- Use CDI to get a handle on a JMS context
- Use the new JMS 2.0 APIs
- Send a message to a queue or topic

- Receive a message via an MDB
- Enlist the messaging operation in a JTA transaction, including XA
- Receive messages via a Message-Driven Bean (MDB)
- Bridge CDI events to JMS
- Dynamically create and destroy a temporary consumer
- Create redelivery, DLQ, and routing policies
- Describe the key techniques for achieving high availability

Contact

Tomitribe
+1 (310) 526-7676
sales@tomitribe.com
www.tomitribe.com