Java SE: Exploiting Modularity and Other New Features

Duration: 2 Days

What you will learn

Java SE: Exploiting Modularity and Other New Features introduces the Java module system and other new features, including JShell, convenience methods, new techniques for working with streams, and managing deprecated APIs. These features were introduced in versions after Java 8, and therefore new to developers migrating to Java 11.

Learn To:

Design applications to take advantage of the module system and its more reliable configuration, improved security and performance, and more easily scalable applications.

Migrate existing applications to a modular applications in a step-by-step manner, choosing which parts of the application to migrate first.

Deal with common problems encountered in migrating an application, including, cyclic dependencies and split packages.

Use services to make modularized applications more robust and easily extensible.

Create multi-release JAR files that can be run on different Java releases.

Use convenience methods to reduce code that seems verbose, inefficient or boilerplate, and increase readability.

Use JShell to quickly run small code experiments and test new APIs.

Benefits To You

By enrolling in this course, you'll learn how to use the module system to design applications with explicit dependencies and encapsulation at the JAR level, ensuring more reliable configuration, improved security and enhanced performance. You'll also get a chance to experiment with new features that ease development. These include convenience methods that make your code more readable and succinct, and JShell, an easy way to test code snippets and APIs.

Audience

J2EE Developer
Java Developers
Java EE Developers
Project Manager

Related Training

Required Prerequisites

Familiarity with JDK 8 features

Familiarity with Java Collections and Enumerators

Familiarity with NetBeans or similar IDE

Familiarity with Object-oriented programming concepts

Suggested Prerequisites

Ability to use classes commonly found in Java programs

Ability to use object-oriented programming techniques

Administer operating systems from the command line

Develop applications using the Java programming languages

Java SE 8 Fundamentals

Java SE 8 Programming

Course Objectives

Identify deprecated APIs and possible alternatives

Swap sub-optimal or tedious coding with convenience methods

Create a modular Java application

Run applications that combine modularized libraries and non-modularized libraries

Create a custom runtime image

Build Multi-release JAR files

Design interfaces which implement methods

Process stream data using new convenience methods

Leverage JShell for fast code experiments

Identify and apply new methods to more conveniently work with collections and arrays

Identify and address common requirements in migrating older applications to modularity

Course Topics

Why Modules?
Module System
Levels of a Typical Application
How Does Java SE 8 Address Maintainability and Reliability?
Classes, Subclasses, Interfaces
Class Level Unit of Reuse (Java SE 8)
Packages
JARs
JAR Files and Distribution Issues

Working with the Module System
Projects Before Modularization
module-info.java: Introduction
Creating a Truly Modular Project
Compiling Modular JAR Files

Accessibility Between Classes
Readability Between Modules
What Is Readable from the competition Module?
The Effects of Exporting

Modular JDK
Modular Development in JDK 9
The JDK
The Modular JDK
Modules in JDK 9
Java SE Modules
The Module Graph of Java SE
The Base Module
Finding the Right Platform Module

Creating Custom Runtime Images
What Is a Custom Runtime Image?
Link Time
Using jlink to Create a Custom Runtime Image
Modules Resolved in a Custom Runtime Image
Advantages of a Custom Runtime Image
JIMAGE Format
Footprint of a Custom Runtime Image
jlink Resolves Transitive Dependencies

Migration
Typical Application
The League Application
Run the Application
The Unnamed Module
Top-Down Migration
Dependencies
Check Dependencies
Typical Application Modularized

Services
Module Dependencies
Service Relationships
Expressing Service Relationships
Using the Service Type in competition
Choosing a Provider Class
Module Dependencies and Services
Designing a Service Type
TeamGameManager Application with Additional Services

Multi-release JAR files
Packaging an Application for Different JDKs
Packaging an Application for Different JDK Versions
The Solution: A Multi-Release JAR file
What Is a Multi-Release JAR File?
Structure of a JAR File
Structure of a Multi-Release JAR File

Search Process in an MRJAR
Creating a Multi-Release JAR File

Private Interface Methods
Private Methods in Interfaces
Java SE 7 Interfaces
Implementing Java SE 7 Interface Methods
Implementing Methods in Interfaces
What About the Problems of Multiple Inheritance?
Inheritance Rules of default Methods
Interfaces Don't Replace Abstract Classes

Enhancements to the Stream API
One More Benefit of Default Methods
Changing a Java Interface
Why Enhance the Stream API?
An Ordered List
takeWhile Provides a Solution
The Effects and Benefits of takeWhile
An Unordered List
filter vs takeWhile

JShell
Has This Happened to You?
A Million Test Classes and Main Methods
JShell Provides a Solution
Comparing Normal Execution with REPL
Getting Started with JShell and REPL
Scratch Variables
Declaring Traditional Variables
Code Snippets

Convenience Methods for Collections
What Are Convenience Methods?
Many Convenience Methods in Java SE 9
Key Collections Interfaces
Overloading the of Convenience Method
Why Overload the of Method?
Growing a Collection
ofEntries Method for Maps
Immutability

Convenience Methods for Arrays
Arrays
Modeling DNA Strands
Working with DNA Strands
Working with DNA Strands by Using a for Loop
Convenience Methods in the Arrays Class
Equating DNA Strands
DNA Subsequences
Equating Subsequences of DNA

Enhanced Deprecations for APIs

What Is Deprecation?
What Is Enhanced Deprecation?
How Do You Deprecate an API?
Using @deprecated
Enhancements to the @Deprecated Annotation in JDK 9
Using the @Deprecated Annotation
Notifications and Warnings
Compiler Deprecation Warnings