The course content of JAVA SE 8

Overview of JAVA Platform

- Illustrating the way, JAVA language achieves platform autonomy
- Comprehension between JAVA platforms: ME, SE and EE
- Evaluation of JAVA libraries, database options and middle-ware
- Describing how JAVA language pursue to evolve

Review: JAVA Syntax and Class

- Designing simple JAVA Classes
- Forging primitive variables
- Using operators
- Implementing if-else and switch statements
- Creating and manipulate strings
- Using if-else and switch statements
- Iterating with the loops: while, do-while, for, enhanced for
- Forging arrays
- Using Java fields, constructors, and methods

Encapsulation and Sub-classing

- Modeling business problems using Java classes
- Using encapsulation in Java class design
- Forging and use Java subclasses
- Preparing classes immutable
- Overloading methods

Overriding Methods, Polymorphism, as well as Static Classes

- Using virtual method invocation
- Using the access levels: private, protected, default, and public
- Overriding methods
JAVA SE 8 Live Online Training Program

- Using the instance of operator to compare object types
- Using upward and downward casts
- Using varargs to specify variable arguments
- Modeling business problems by using the static keyword
- Implementing the singleton design pattern

Interfaces and Lambda Expressions

- Defining What JAVA interface is?
- Extending an interface
- Defaulting methods
- Anonymous inner classes
- Choosing between interface inheritance and class inheritance
- Defining a Lambda Expression

Abstract and Nested Classes

- Constructing abstract Java classes and subclasses
- Designing general-purpose base classes by using abstract classes
- Distinguish between top-level and nested classes
- Applying final keyword in Java

Collections and Generics

- Forging a customize generic class
- Implementing an Array List
- Using the type inference diamond to create an object
- Creating a collection by using generics
- Implementing a HashMap
- Implementation of a Deque
- Implementation of a TreeSet
- Ordering collections

Collections Streams and Filters

- Illustrating the builder pattern
- Iterating through a collection using lambda syntax
- Illustrating the Stream interface
- Filtering a collection using lambda expressions
• Chaining multiple methods together
• Describing pipelines in terms of lambdas and collections
• Calling an existing method using a method reference

What is Lambda Built-in Functional Interfaces?

• Listing the built-in interfaces included in java.util.function
• Using primitive versions of base interfaces
• Using binary versions of base interfaces
• Core interfaces - Predicate, Consumer, Function, Supplier

Lambda Operations

• Defining the types of stream operations
• Defining the Optional class
• Extracting data from an object using map
• Sorting a stream
• Describing lazy processing
• Saving results to a collection using the collect method
• Grouping and partition data using the Collectors class

About Exceptions and Assertions

• Using the try and throw statements
• Using the catch, multi-catch, and finally clauses
• Describing the purpose of Java exceptions
• Auto-close resources with a try-with-resources statement
• Recognizing common exception classes and categories
• Forging custom exceptions
• Testing invariants by using assertions

Java Date/Time API

• Creating and managing time-based events
• Creating and managing date-based events
• Combining the date and time into a single object
• Managing changes resulting from daylight savings
• Describing and create timestamps, periods and durations
Working with the dates and times across the time zones
Applying formatting to local and zoned dates and times

I/O Fundamentals

- Reading and writing data from the console
- Using the streams in reading and writing the files
- Describing the basics of input and output in Java
- Writing and reading objects using serialization

File I/O (NIO.2)

- Use the Path interface to operate on file and directory paths
- Use the Files class to check, delete, copy, or move a file or directory
- Using the Stream API with NIO2

Concurrency

- Creating worker threads by using Runnable and Callable
- Using an Executor Service to concurrently execute tasks
- Defining operating system task scheduling
- Identifying the potential threading problems
- Using the synchronized and concurrent atomic in managing atomicity
- Using monitor locks to control the order of the thread execution
- Using the java.util.concurrent collections

Fork-Join Framework

- Need for Fork-Join
- Parallelism
- Recursive Task
- Work stealing

Parallel Streams

- Analyzing the key characteristics of streams
- Defining the way to make a stream pipeline execute in parallel
- List the key assumptions needed to use a parallel pipeline
• Describing reduction
• Defining why reduction requires an associative function
• Calculating the value using reduce
• Listing the key performance considerations for the parallel streams
• Describing the process for decomposing and then merging the work

Database Applications with JDBC

• Connecting to a database using the JDBC driver
• Submitting queries and get results from the database
• Describing the layout of the JDBC API
• Performing CRUD operations using JDBC API
• Specifying the JDBC driver information externally

Localization

• Defining what a locale represents
• Defining the advantages of localizing an application
• Reading and setting the locale by using the Locale object
• Building a resource bundle for each locale
• Changing the locale for a resource bundle
• Calling a resource bundle from an application