# **Java Stack: Building Enterprise applications**

Total duration: 25 days

## Course Duration:

**25 days [8 hours per day] including Theory and Lab**

## Pre-requisites:

## Any Programming language

## Objectives:

* Understand object-oriented programming using Java
* Apply SOLID design principles while writing code
* Understand how to write code to interact with relational database system
* Understand writing concurrent programming
* Understand using data structures
* Understand how to write web applications
* Understand Object Relational Mapping
* Understand how to use JPA and Spring for building enterprise applications
* Understand using Spring BOOT for building RESTful web services

## Course Contents

**Part 1: 6 days**

**Day 1**

* Introduction
  + Features of Java
  + Java Architecture
  + JVM, JRE, JDK
* OO Programming in Java
  + Object
  + Class
  + Class Specification
  + Characteristics of an Object
  + Basic Principles of Object Orientation
  + Abstraction
  + Encapsulation
  + Modularity
  + Hierarchy
  + Polymorphism and overloading/overriding
  + Constructors
  + Static attributes and Static Methods
  + Packaging in Java
* Relationships between Classes
  + - Generalization and Specialization
    - Realization relationship

**Day 2**

* Interfaces
  + What is an interface?
  + Writing loosely coupled applications using Interface
* Inner Classes and Anonymous classes
* Error Handling with Exceptions
  + - Basic exceptions
    - Exception arguments
    - Catching an exception
    - The try block
    - Exception handlers
    - Creating your own exceptions
    - Catching any exception
    - Re-throwing an exception
    - Standard Java exceptions
    - Exception guidelines

**Day 3**

* Collections
  + - Sorting and searching Lists Advanced Array Operations
    - Sorting arrays and the Equals Implementation
    - Containers
    - Filling Containers
    - Disadvantages of Object Containers
    - Making a type conscious Array List
    - Parameterized types
    - Iterators
    - Container Taxanomies
    - Collection Functionality
    - List Functionality
    - Stack and Queue Implementations
    - Set Functionality
    - Sorted Set
    - Map functionality
    - Generic Collections

**Day 4**

* Threads
  + - Describe a thread
    - Create separate threads, controlling the code and data that are used by
    - that thread
    - Use the keyword synchronized to protect data from corruption
    - Use wait() and notify() to communicate between threads
    - Multithreaded Programs
    - Synchronizing Threads
    - Concurrency utilities (java.util.concurrent)
    - Callable and Future
    - Executors and ExecutorService
    - Lock APIs
* JDBC
  + - Introduction
    - DB Drivers
    - Driver Manager
    - Connection
    - Statements and prepared statements
    - Result sets and Meta Data
* SOLID Design principles
* Unit testing using Junit
* Mockito
* Using GIT for Source control
* Checkstyle for Coding Conventions
* PMD for checking coding standards
* Java 8 Concurrency
  + Fork Join Framework
  + CompletableFuture

**Day 5**

* Java 8 Lambda expressions
  + Lambdas: intro, motivation, and big idea
  + Lambda expression interpretation
  + Most basic form of lambdas
  + Type inferencing
  + Expression for body
  + Omitting parens
  + Using effectively final variables
  + @FunctionalInterface
  + Default and static methods in interface
  + Method references
  + java.util.function package
  + Lambda building blocks in java.util.function: overview
  + Predicate
  + Function and BiFunction
  + Consumer
  + Supplier
  + High Order functions, Function Composition and Currying
  + Variable scoping: Local variables, instance variables, "this"
  + Method references: details -- Class::staticMethod, variable::instanceMethod Class::instanceMethod, Class::new
  + New features in Java 8 interfaces: Default methods (also called "virtual extension methods" or "defender methods") and static methods
* Java 8 Streams
  + Overview of Streams
  + Building Streams
  + Outputting Streams into arrays or Lists
  + Core Stream methods:
  + Intermediary methods: map, filter, skip, limit, flatMap
  + Terminal methods: reduce, forEach, collect, count
  + Lazy evaluation and short-circuit operations
  + Parallel Streams

**Day 6**

* Introduction to Web development using Servlet Technology
  + Servlet API
  + Servlet Life cycle
  + HttpServletRequest and HttpServletResponse
  + JSP and JSTL
  + Server-side redirection using RequestDispatcher
  + Client-side redirection using sendRedirect
  + Session handling
  + HttpSession
  + Listeners and Filters
  + MVC Architecture pattern

**Part 2: 2 days**

**Day 7**

* *Introduction to ORM*
  + Issues with Persistence layers and Object-Relational Mapping (ORM)
  + Hibernate Overview and Benefits
  + Hibernate architecture overview
  + POJO (Plain Old Java Object) Based Mapping

*Getting started with JPA/Hibernate*

* + Overview of the Hibernate
  + Configuring JPA/Hibernate
  + Persistence.xml file
  + EntityManagerFactory,EntityManager
  + Connection properties, Database dialect
  + Mapping a Class
  + Persistent Entity Class
  + Primary keys: Id property, Generated Id

Querying

* + Inserting and Updating Entities
  + JPQL
  + The Query Interface
  + Creating and working with queries
  + Named Queries, Projection Queries, Aggregate Queries

**Day 8**

*The Persistence Lifecycle*

* + Transaction Overview and Transactions in Hibernate
  + Hibernate Transaction API (in Managed and Non-managed Environments)
  + The lifecycle of managed objects
  + Persistent, transient, and detached objects
  + Synchronization to the Database
  + The Session as cache

Relationships

* + Object Relationship Overview
  + Mapping Collections of Value Objects
  + Entity Relationships: 1-N, N-1, N-N, 1-1
  + Mapping Entity Relationships
  + Uni and Bi-directional Relationships
  + The Relationship "inverse"
  + Cascading Over Relationships
  + Queries Across Relationships (Lazy and Eager)

*Inheritance Mapping*

* + Entity Inheritance with Hibernate
  + Table-per-class mapping
  + Table per Subclass mapping
  + Table per Concrete Class mapping

**Part 3: 7 days**

**Day 9**

Introduction to Spring

* + Overview of Spring Technology
  + Motivation for Spring, Spring Architecture
  + The Spring Framework
  + Maven and Spring
  + Spring Introduction
  + Declaring and Managing Beans
  + ApplicationContexts - The Spring Container
  + XML and @Component/@Named Config
  + Dependencies and Dependency Injection (DI)
  + Examining Dependencies
  + Dependency Inversion / Dependency Injection (DI)
  + DI in Spring - XML and @Autowired

*Configuration in Depth*

* + Java Based Configuration (@Configuration)
  + Overview, @Configuration, @Bean
  + Dependency Injection
  + Resolving Dependencies
  + Integrating Configuration Types
  + XML and @Component Pros/Cons
  + @Configuration Pros/Cons
  + Choosing a Configuration Style
  + Integrating with @Import and <import>
  + Bean Scope and Lifecycle
  + Singleton, Prototype, and Other Scopes
  + Configuring Scope
  + Bean Lifecycle / Callbacks

**Day 10**

* Spring Web MVC
  + Introduction to Spring Web MVC framework
  + Dispatcher Servlet and ApplicationContext
  + Controller and Handler mappings in Spring
  + Spring’s Form tag library
  + PropertyEditors
  + Validation in Spring MVC
  + Using the Message Source for error messages
  + View Resolvers
* Implementing Enterprise Information Connectivity [Integrating JPA with Spring]
  + The LocalContainerEntityManagerFactoryBean
  + Different states of an Object: Transient state, persistent/ managed state, Detached state and removed state.
  + Persisting an entity
  + Executing queries using JPQL, Criteria API and Native queries
  + Named Queries
  + Locking and Concurrent access
  + Transaction management
  + The spring transaction abstraction.
  + Transaction strategies.
  + Declarative transaction management
  + The tx namespace

**Day 11**

* **Spring Boot**
  + Introduction to Spring Boot
  + Bootstrapping Spring boot application
  + Starting Annotations of Spring Boot
    - @SpringBootApplication
    - @Repository
    - @Service
    - @Configuration
    - @Bean
  + Spring Profiles
* **Spring Data JPA**
  + - Spring data JPA CRUD
      * Repository, CrudRepository and JpaRepository
      * Query methods
        + Query generation using method name

find…By, read…By, query…By, count…By, and get…By.

* + - * + Using @Query annotation
        + Named Queries
      * Modifying queries
      * Query Hints
      * Using SpEL Expressions
        + Using #{#entityName}
        + Accessing arguments
        + Using wildcard
        + Sanitizing input values
      * Projections
        + Closed and open projections
        + Projection interface using default method for custom logic
        + Dynamic projections
      * Pagination and Sorting
        + PagingAndSortingRepository
        + Pageable, Page, Sort and PageRequest
      * Passing method parameters to Query methods
      * Specifications
        + JpaSpecificationExecutor
      * Query by Example
        + Example
        + Example Matchers
      * JPA Entity graphs
        + Benefits of using Entity Graphs
        + @EntityGraph
        + @NamedEntityGraph
      * Spring Data — Envers
        + @Audited
        + @AuditTable
        + RevisionRepository
        + Enable EnversRevisionRepositoryFactoryBean
        + Querying data
    - Unit testing
      * Spring data jpa
  + Caching
    - Register cache engine with spring boot
    - Enable caching on spring managed beans
    - @EnableChaching
    - @Cachable
    - @Caching
      * @CacheEvict
* Spring Boot Actuator
  + - Monitoring
    - Built-in production ready endpoints
    - Monitoring with Prometheus and Grafana

**Day 12**

* **RESTful Web Services**
* What is REST
* HTTP Methods and Status Codes
  + GET, POST, PUT, PATCH, DELETE
  + PUT vs PATCH vs JSON PATCH
* Richardson Maturity Model
* Resources and URI
  + REST Resource naming conventions
  + Use Nouns to represent resources
    - Document
    - Collection
    - Store [ client managed resource repository]
    - Controller [ Procedural concepts like checkout, play]
  + Nested Resources
    - Why?
    - Why not?
  + Good practices in resource naming conventions
    - Use forward slash to indicate hierarchical relationships
    - Do not use trailing slash in URIs
    - Use hyphens (-) to improve readability of URIs
    - Avoid using underscore (\_)
    - Use lower case letters in URIs
    - Do not use CRUD functions in URIs
    - Use query component to filter URI collection
  + Fine grained CRUD resources versus Coarse Grained resources
  + Pagination, Filtering and Sorting
  + Versioning
* Guiding Principles of REST
  + Client-server
  + Stateless
  + Cacheable
  + Uniform Interface
  + Layered System

**Day 13**

* RESTful Web Services using Spring BOOT
  + Creating REST controllers
    - @RestController
    - @RequestMapping
    - CRUD Mapping: @GetMapping(), @PostMapping(), @PutMapping(), etc
    - @ResponseBody and @RequestBody
    - @ResponseStatus
    - Exception Handling
      * Using Own exception: customize exception handling
      * @ControllerAdvice and @ExceptionHandler
  + Using POSTMAN
  + RestTemplate vs WebClient
  + Content Negotiation
    - Content Negotiation: XML/JSON or your own content type
    - Custom Message Convertor
    - Creating your own content-type
  + Using Swagger to document RESTful web services

**Day 14**

* Spring Boot: Unit Testing Rest Services
  + Unit testing the GET Rest Service
    - @RunsWith: using SpringJUnit4ClassRunner to launch Spring TestContext Framework
    - @WebMvcTest: Unit testing Spring MVC Framework
    - MockMvc and MockBean: adding mocks to Spring ApplicationContext
    - Mockito: Mocking behaviour
* Spring HATEOAS
  + Building Hypermedia driven RESTful web service
    - RepresentationModel
    - Creating links using Link object
    - WebMvcLinkBuilder – which simplifies building URIs
      * The linkTo() method
      * The slash() method
      * The withSelfMethod()
    - Relations
    - HAL
      * Hyperlink between resources in API
      * HAL Browser
* Spring Data REST
  + - Build hypermedia-driven REST web services with spring data repository
* Spring Security
  + Spring Security Architecture
  + Getting Spring Security
  + Web Security Java Configuration
    - @EnableWebSecurity
    - WebSecurityConfigurerAdapter
    - UserDetails and UserDetailService
    - InMemoryUserDetailsManager
    - AuthenticationProvider
      * DaoAuthenticationProvider
        + Custom UserDetails, UserDetailService and UserRepository
        + NoOpPasswordEncoder and BCryptPasswordEncoder
    - AuthenticationManagerBuilder
      * inMemoryAuthentication
      * jdbcAuthentication
    - HttpSecurity
      * CSRF
      * formLogin(), loginPage()
      * Authorize Requests
        + antMatchers() , anyRequest()
        + permitAll() and authenticated()
        + hasRole()
      * logout
        + Invalidate HttpSession and Clear authentication info
        + logoutRequestMatcher [ AntPathRequestMatcher ]
        + logoutSuccessUrl
      * SPRING\_SECURITY\_LAST\_EXCEPTION
    - Method Security
      * EnableGlobalMethodSecurity
        + @EnableGlobalMethodSecurity(prePostEnabled = true)
        + @PreAuthorize and @PostAuthorize
      * GlobalMethodSecurityConfiguration
        + MethodSecurityExpressionHander
        + Custom Permission Evaluator
        + JWT

**Day 15**

* Building Microservices with Spring Boot **Introduction** with example
  + Spring Cloud and Netflix OSS
    - Service Discovery using Eureka
      * Registering with Discovery Service
    - HTTP clients with Feign
    - Api Gateway
* Protocol Buffers
  + What and Why?
  + Define message structures with .proto files
  + Protocol buffers with Java
  + Protobuf in Spring REST API

## **Web Development using JavaScript Stack**

## **JavaScript, TypeScript, MongoDB, NodeJS and React**

## Course Duration:

**10 days [8 Hours /day] – including theory and lab**

## Objectives:

* Understand Functional style of programming using JavaScript
* Get familiar with NodeJS and Angular framework
* Learn how Node.js is architected to allow high scalability with asynchronous code.
* Use stream I/O to efficiently serve the web pages
* Create modules to organize the server
* Test the reliability of the application with unit tests
* Interface to a MongoDB database and a web service
* Get up to speed with TypeScript
* Implement single-page applications
* Use various Angular features like directives, components and services
* Implement one-way as well as Two-Way Data Binding
* Handle Angular forms
* Use Angular modules and optimize apps
* Learn & Implement Dependency Injection
* Change pages with Routing
* Understand & use Observables
* Implement a functional web application using Angular
* Running tests with CLI and testing dependencies

## High Level Course Details:

* Functional style of Programming
* ES2015
* Unit Testing using Jasmine
* Node JS
* MongoDB
* TypeScript and Webpack
* Angular Components and Directives
* Angular Forms and services
* Reactive Programming
* Routing and Guards
* Angular Design patterns
* E2E testing using Protractor

## Detailed Course Outline:

**PART 1: 3 days**

**Day 16**

### HTML, CSS and JavaScript

* Displaying the web with HTML
* HTML basic tags
* Tables, list, Hyperlink
* HTML Forms
* Styling with CSS
* CSS Basic Syntax
* CSS Selectors
* Styling background, images, list and fonts
* CSS3 Box Model
* Using Bootstrap libraries for Responsive Web design
* Grammar of JavaScript
* Statements
* Commands
* Types of Data
* Variables
* Conditional Statements and handling repetitive tasks using loops
* Event Handling
* Understanding Document Object Model
* Using jQuery for DOM selection, traversal and manipulation
* AJAX, Using jQuery to make AJAX calls

**Day 17**

### Functional style of Programming

* Functions in JavaScript
  + Functions as First-Class Objects
  + Closures
  + High Order functions
    - Writing high order functions
    - Currying
    - Implementing Map, reduce and filter functionalities using high order functions

### ES2015

* Arrow functions – A short-hand version of an anonymous function.
* Block-level scope – ES6 now supports scoping variables to blocks (if, for, while, etc.) using the let keyword.
* Classes – ES6 classes provide a way to encapsulate and extend code.
* Constants – You can now define constants in ES6 code using the const keyword.
* Default parameters – Ever wished that a function parameter could be assigned a default value? You can do that now in ES6.
* Modules – Provides a modular way of organizing and loading code.
* Promises – Used with async operations.

**Day 18**

* Node.js
* • Installing Node.js
* • Node's Event Loop
* • Alternatives to Node.js
* • Writing asynchronous code
* • Modularizing code
* o Understanding built-in modules
* o Techniques for modularizing JavaScirpt code
* o Using require() to modularize application code
* o Using npm for third-party modules
* • Handling Exceptions
* • Events and Streams
* o Understanding Events
* o EventEmitter class
* o Understanding Streams
* o Reading and writing streams
* o Using pipe()
* • Node.js and the web
* o Handling web requests
* o Building a web server
* o Understanding the need for web sockets
* o Realtime interaction using socket.io
* • Node JS External Process
* o Launching commands and child processes, Sending and receiving data from a
* child process, Sending signals to and then terminating a child process
* Node.js
* • Installing Node.js
* • Node's Event Loop
* • Alternatives to Node.js
* • Writing asynchronous code
* • Modularizing code
* o Understanding built-in modules
* o Techniques for modularizing JavaScirpt code
* o Using require() to modularize application code
* o Using npm for third-party modules
* • Handling Exceptions
* • Events and Streams
* o Understanding Events
* o EventEmitter class
* o Understanding Streams
* o Reading and writing streams
* o Using pipe()
* • Node.js and the web
* o Handling web requests
* o Building a web server
* o Understanding the need for web sockets
* o Realtime interaction using socket.io
* • Node JS External Process
* o Launching commands and child processes, Sending and receiving data from a
* child process, Sending signals to and then terminating a child process
* Node.js
* • Installing Node.js
* • Node's Event Loop
* • Alternatives to Node.js
* • Writing asynchronous code
* • Modularizing code
* o Understanding built-in modules
* o Techniques for modularizing JavaScirpt code
* o Using require() to modularize application code
* o Using npm for third-party modules
* • Handling Exceptions
* • Events and Streams
* o Understanding Events
* o EventEmitter class
* o Understanding Streams
* o Reading and writing streams
* o Using pipe()
* • Node.js and the web
* o Handling web requests
* o Building a web server
* o Understanding the need for web sockets
* o Realtime interaction using socket.io
* • Node JS External Process
* o Launching commands and child processes, Sending and receiving data from a
* child process, Sending signals to and then terminating a child process

### Jasmine- JavaScript unit testing

* Introduction to Jasmine
* Get Jasmine
* The Spec and SpecRunner
* Matchers
* Create your own matchers
* Before and After
* Spies
* Testing Asynchronous code

**Part 2: 3 days**

**Day 19**

### Node.js

* Installing Node.js
* Node's Event Loop
* Alternatives to Node.js
* Writing asynchronous code
* Modularizing code
* Understanding built-in modules
* Techniques for modularizing JavaScirpt code
* Using require() to modularize application code
* Using npm for third-party modules
* Handling Exceptions
* Events and Streams
* Understanding Events
* EventEmitter class
* Understanding Streams
* Reading and writing streams
* Using pipe()
* HTTP Server with Node.js and Core http Module
* Node.js, Web Apps and http Core Module
* Node.js Hello World HTTP Server
* Node.js Hello World HTTP Server Demo
* Unit testing with Mocha

**Day 20**

**MongoDB**

* What is MongoDB?
* What Is Meant by NoSQL?
* Why to use MongoDB?
* Common Terms in MongoDB
  + \_id
  + Collection
  + Cursor
  + Database
  + Document
  + Field
  + JSON
* Data Modelling
* Difference between MongoDB & RDBMS
* Download & Install MongoDB
* MongoDB Configuration, Import, and Export
* Creating a database
* Creating a collection
* Adding documents using insert() command
* MongoDB ObjectId()
* Reading documents / Performing Queries/ Using $regex operator for Pattern matching

**Day 21**

* MongoDB Query Modifications using limit(), sort()
* MongoDB Count() & remove() function
* MongoDB Update() Document / Updating Multiple Values
* MongoDB Indexing, Monitoring & Backup
* User Management
* Connecting to MongoDB from NodeJS
* Mongoose => MongoDB Object modelling tool
* NPM install mongoose
* Defining Schemas
* Creating a model
* Schema types
* Methods and query helpers
* Working with related documents

REST API with MongoDB and Node.js using Express Framework

### **Part 3: 4 days**

**Day 22**

### TypeScript Introduction

* + Types as in TypeScript
  + Enums
  + Return types
  + Interfaces
  + Optional arguments
  + Classes
  + Working with other libraries
  + Decorators

### Webpack

* + Getting started
  + Defining a config file
  + Watchmode
  + Webpack Dev Server
  + Webpack loaders and preloaders
    - Using babel with webpack
    - JSHint with webpack
  + Production Webpack bundles

**Day 23**

### ReactJS

Introduction

* What is React?
* Real World SPAs & React Web Apps
* React Alternatives

React Components

* Component basics
* Component architecture
* Virtual DOM
* Splitting app into components
* Functional components
* Component Implementation
* Component Composition
* Composition Implementation
* Lifecycle Methods
* JSX
* React State and Props
  + Manging Data in React
  + State and Props Implementation
* React Event Handling
* Working with Forms and Events

Testing React

* Introduction to JEST
* React Testing Library
* Rendering a component
* Selecting elements
* Search Types
  + - getByText, getByRole, getByLabelText, getByPlaceholderText, getByAltText, getByDisplayValue
    - Search Variants: queryByXXX and findByXXX
    - Using **container** to query for rendered elements
* Using screen.debug()
* Using RTL’s Assertive functions
* Fire Event and User Event
* Mocking callbacks and Testing async
* Code coverage
* E2E testing with Cypress

Styling

* Using styled components
* React Spectrum
  + Layout
  + Styling
  + Theming

**Day 24**

React-Router

* Routing and SPAs
* Setting up links
* Rendering components for Routes
* Using Routing related props
* Passing and extracting Route parameters
* Navigating programmatically
* Redirecting Requests

Context

* Passing data through the component tree without having to pass props down manually at every level
* React.createContext
* Context.Provider
* Context.Consumer

HTTP/ Connecting to REST endpoints

* Fetching data via Ajax
* Rendering fetched data to the screen
* Posting data via Ajax
* Creating and using Axios / fetch

High Order Components

* Props proxy
* Inheritance Inversion

Error boundaries

* Use static getDerivedStateFromError() to render a fallback UI after an error has been thrown.
* Use componentDidCatch() to log error information.

Refs and DOM

* Creating Refs
* Forwarding Refs

React Hooks

* useState
* useReducer
* useEffect
* useCallback
* useMemo
* useRef
* useContext

**Day 25**

Redux

* Problems of Flux pattern
* Building blocks in Redux
* Action
* Action Creators
* The store
* The reducers
* Combine reducers
* Views: smart and dumb view
* React-Redux Bindings
* The root component
* The data flow in Redux
* Testing
* Testing Action creators
* Testing Reducers
* Mock Store using redux-mock-store
* Mock HTTP requests using fetch-mock

Middleware

* Using Middleware
* Creating Custom Middleware
* Creating a Logger Middleware
* Configure Redux DevTools Extension
* Redux Thunk
  + Handling Asynchronous Redux actions
* Redux-Saga
  + Make application side effects easier to handle
  + Using Redux saga helper functions: takeEvery(), takeLatest(), put(), call()
  + Running effects in parallel: all() and race()
* Using Redux Toolkit

React’s Performance

* The shouldComponentUpdate()
* PureComponent
* React memo
* Binding in Constructors vs Arrow functions
* Avoid binding when rendering
* Using proper key property while rendering lists
* React Fragments
* Debouncing event action
* React.lazy for code-splitting
* Using Suspense for Data fetching
* Using Web workers for CPU extensive tasks
* Core Web vitals
  + LCP, FID, CLS, TTFB
  + The React Profiler API
  + The Interaction Tracing API
    - Using React DevTools
  + User Timing API
    - Using LightHouse
    - Using Chrome DevTools Performance panel