Maslak Mahallesi, Maslak Meydan Sk. No:5. Spring Giz Plaza. Maslak/İstanbul
+90 212 282 7700 - info@bilginc.com

Fast Track to Java

Learn via: **Classroom / Virtual Classroom / Online**

Duration: **4 Day**

https://bilginc.com/en/training/fast-track-to-java-5595-training/

Overview

This is an intensive course designed to introduce software developers to the Java ecosystem in general and the Java 9 language in particular. All four programming styles supported by Java are covered in depth, but with a particular emphasis on the functional style and its growing importance to both parallel and cloud computing.

The course is aimed primarily at three groups. Firstly experienced coders who will be moving into Java from other languages, secondly developers who need an update to the new features in Java 8 / 9 and finally graduates looking to consolidate knowledge gained through academia

In addition to the core language topics, the course also covers a variety of the skills required on modern Java / JEE projects. These include Reactive Programming (via Project Reactor), Test Driven Development (via JUnit / Mockito) and Behaviour Driven Development (using Cucumber).

Prerequisites

Delegates must be proficient programmers, ideally with several years commercial programming experience

What You Will Learn

- Learn Java from the basics right up to the latest Java 9 features
- Understand when to apply all supported programming styles
- Learn how to use TDD and BDD to test your code
- Become comfortable applying functional and reactive programming techniques

Outline

Introducing the Java Landscape

- Origins and goals of the Java language
- Bytecode and the Java Virtual Machine
- Distinguishing between JME and Android
- Where Microservices and the JEE diverge
- Popular languages on the Java platform

Essential JVM Concepts

- Types, packages and archives (JAR/WAR/EAR)
- References and tuning Garbage Collection
- Hierarchical class loading and Agents
- The CLASSPATH environment variable
- The new Module System in Java 9

Basic Java Programming

- Primitive types and literal values
- Java syntax for binary and numeric literals
- The difference between reference and value types

- Converting between strings and numerical types
- Parsing console input with the Scanner class
- Pretty-printing with the Formatter class
- Performing iteration and selection in Java
- Support for strings in switch statements
- Equality with primitive and reference types
- Creating and manipulating arrays
- The costs of string concatenation

Object Oriented Development - Part 1

- Creating basic Java classes
- Choosing accessibility levels
- Inheriting from a base class
- Overloading and overriding methods
- Comparisons using the instanceof operator
- Comparisons using java.lang.Class objects
- Creating abstract and final classes

Object Oriented Development - Part 2

- Writing appropriate class constructors
- Private constructors and singletons
- Static and instance initialization blocks
- Top down class and object initialization
- Declaring and implementing interfaces
- Using inner and anonymous classes
- Implementing equals and hashCode
- Cloning and copy constructors

Enumerations

- Why Java historically lacked enum support
- The Typesafe Enumeration Design Pattern
- How Java 5 baked the pattern into the language
- Extending enums with new members
- Helper methods added to enum types
- Collections which support enums

Annotations and Meta-Programming

- Adding metadata to Java code
- The advantages of annotations
- Annotations verses configuration files
- Declaring Java 5 annotation types
- Understanding meta-annotations
- Adding methods to annotation types
- Defining default values for methods
- Discovering annotations with reflection
- Writing annotation processors in Java 6
- Enhancements to annotations in Java 8

Functional Programming with Method References and Lambdas

- Overview of the key concepts of Functional Programming
- The concept of 'method references' in the language and JVM
- Creating references to methods, static methods and constructors
- Understanding when and why to use lambda expressions
- The various syntax options when declaring lambdas
- Making use of the @FunctionalInterface annotation
- Lexical scoping and 'effectively final' variables
- Support for type inference of parameters in lambdas
- Best practices when declaring and using lambdas

Introducing the Collections API

- Introducing lists, sets and maps
- Using iterators and enumerations
- Choosing the most efficient collections
- Specialized collections in other packages
- Collections designed for concurrent access
- Open source alternatives to regular collections

Functional Programming With Collections and Streams

- Introducing the java.util.stream.Stream interface
- Different ways of creating streams of objects
- How Optional helps eliminate null pointer exceptions
- Using parallel and sequential to switch processing model
- Performing filtering and mapping on a streams content
- Using reduce to compute a final value from a stream
- Understanding the purpose and value of flatMap
- The collect method and its many different applications
- Enhancements to the streams library in Java 9
- Support for Reactive Programming in Java 9
- Taking Rx further via Project Reactor

Exception Handling in Java

- Introducing errors, runtime exceptions and checked exceptions
- Special consideration for exceptions in constructors and finalizers
- Implementing an effective exception handling strategy
- Using finally blocks properly to perform 'clean-up' tasks
- Java 7 improvements to the try...catch syntax
- The try-with-resources syntax introduced in Java 7

Generics in Java

- Introducing Type Parameters and generic code
- Understanding reified vs. non-reified Generics
- Generics is a compile time feature in Java
- Declaring generic classes and methods
- Type inference when creating generic types
- Creating your own generic collection classes
- Support for Generics in the Reflection API
- Using the wildcard type in utility methods
- Defining constraints with bounded wildcards
- Making sense of upper and lower bounds
- Common confusions with wildcards
- How compilers perform Type Erasure

The Date and Time API

- How JSR310 evolved from the Joda Time library
- Working with instants, clocks and time zones
- Performing calculations with dates and times
- Using adjusters to make common changes to dates
- Parsing and formatting dates and times
- Combining the API with legacy code

TDD in Java with JUnit and Mockito

- Defining your intent through tests
- Writing just enough code to pass
- Adding tests and refining the code
- Testing up to the point of boredom
- Triangulating on hard problems
- Moving up and down the gears
- You aren't going to need it (YAGNI)
- Why encapsulated dependencies are bad
- Using DI to isolate dependencies
- Creating and injecting Test Doubles
- Situations where TDD will not work

Behaviour Driven Development in Java with Cucumber

- How BDD evolved from Test Driven Development
- Writing scenarios using the Gherkin syntax
- Moving shared steps into a background section
- Embedding data tables within scenarios
- Setting up JUnit to run and configure Cucumber
- Creating a class to hold step definitions
- Writing step definitions via annotations in Java 6
- Writing step definitions via lambdas in Java 8

- Matching to the content of steps via regex
- Using sub-matches to capture input values
- How cucumber converts and tokenizes inputs
- Reading and extracting values from data tables