**Service Virtualization**

**Training Agenda**

**Duration:** 4 days

**Prerequisite :** Understanding of programming language is needed, and having work experience with Java is preferred. Theoretical knowledge of functional programming is an added advantage.

**Day 1**

**Functional Programming With Java**

* Interface Static Methods
* Default Methods
* Functional Interfaces
* Lambda Syntax
* Using Lambdas
* Method References
* Choosing Functional Interfaces
* Closures
* Lambda Compositions

**Functional Programming With Java**

* Working with JUnit 5
* Creating Testcases
* JUnit 5 Annotations
* JUnit Assertions
* Assumptions
* Using @Test in Junit5
* Using Annotations - @BeforeAll and @AfterAll
* TestFixtures with @BeforeEach and @AfterEach
* Testing Exceptions using assertThrows

**Day 2**

**Module 1: Mockito Introduction**

* What is mocking?
  + Why should we mock?
* Project creation
* Mockito installation
  + Download the JAR
  + With build tools
    - Maven
* Base code to test
* Adding behavior
* Verifying behavior
  + Verify that method has been called
  + Verify that method has been called n times
  + Verify method call order
  + Verification with timeout
* Throwing exceptions
* Shorthand mock creation
* Mocking void returning methods
* Mocking real objects: @Spy

**Module 2: Test-Driven Development with Mockito**

* Introduction
* Test Driven Development
* Creating a project
  + Dependencies
* Implementing behaviour-driven development (BDD) with Mockito to improve collaboration and test clarity.

**Module 3: Mockito Initmocks Example**

* Introduction
* Creating a project
  + Dependencies
* Init Mocks
  + Using Mockito.mock()
  + MockitoAnnotations initMocks()
    - Inject Mocks
  + MockitoJUnitRunner
  + MockitoRule

**Module 4: Mockito Maven Dependency Example**

* Introduction
* Creating a project
* Adding dependencies
* Testing

**Module 5: Writing JUnit Test Cases Using Mockito**

* Introduction
* Creating a project
* Dependencies
* Verify interactions.
* Stub method calls
* Spy
* InjectMocks
* Argument Matchers

**Day 3**

**Module 6: Mockito: How to mock void method call**

* Introduction
* Creating a project
  + Dependencies
* Stub
  + doThrow()
  + doAnswer()
  + doNothing()
    - Stubbing consecutive calls on a void method
    - When you spy real objects, and you want the void method to do nothing.

**Module 7: Spring Test Mock Example**

* Introduction
* Creating a project
  + Dependencies
* Applying Mockito for testing microservices and distributed systems
* Mocking and verifying interactions with external dependencies and APIs.

**Module 8: Mockito Captor Example**

* Introduction
* Creating a project
  + Dependencies
* ArgumentCaptor class
* Methods
  + public T capture()
  + public T getValue()
  + public java.util.List<T> getAllValues()
* Captor annotation
* Code
  + Simple Code

**Module 9: Mockito ThenReturn Example**

* Introduction
* Creating a project
  + Dependencies
* thenReturn
* Code

**Day 4**

**Module 10: What is Service Virtualization**

* Defining Service Virtualization
* Understanding the concept of service virtualization and its importance in modern software development.
* Service Virtualization in Action
* Case Study example
* Seeing How Service Virtualization Differs
* from Other Types of Virtualizations
* Exploring Where Service Virtualization Can Add Value
* Testing
* Development
* Non-production usage
* Benefits of Service Virtualization
* Reducing costs
* Improving productivity
* Reducing risk
* Increasing quality

**Module :11 Mocking and Virtualization**

* API Mocking
* Difference between Virtualization and Mocking
* API Virtualization

**Module:12 Implementing Service Virtualization Using WireMock**

* Setting up WireMock
* Download and Installation
* Running as a Standalone Process
* Running in Docker
* Configuration
* HTTPS
* Multi-domain Mocking
* Using Junit and WireMock
* Java configuration
* Running without the HTTP Server
* Stubbing and Verifying
* Request Matching
* Response Templating
* Simulating Faults
* Stateful Behaviour
* Proxying
* Verifying
* Record and Playback