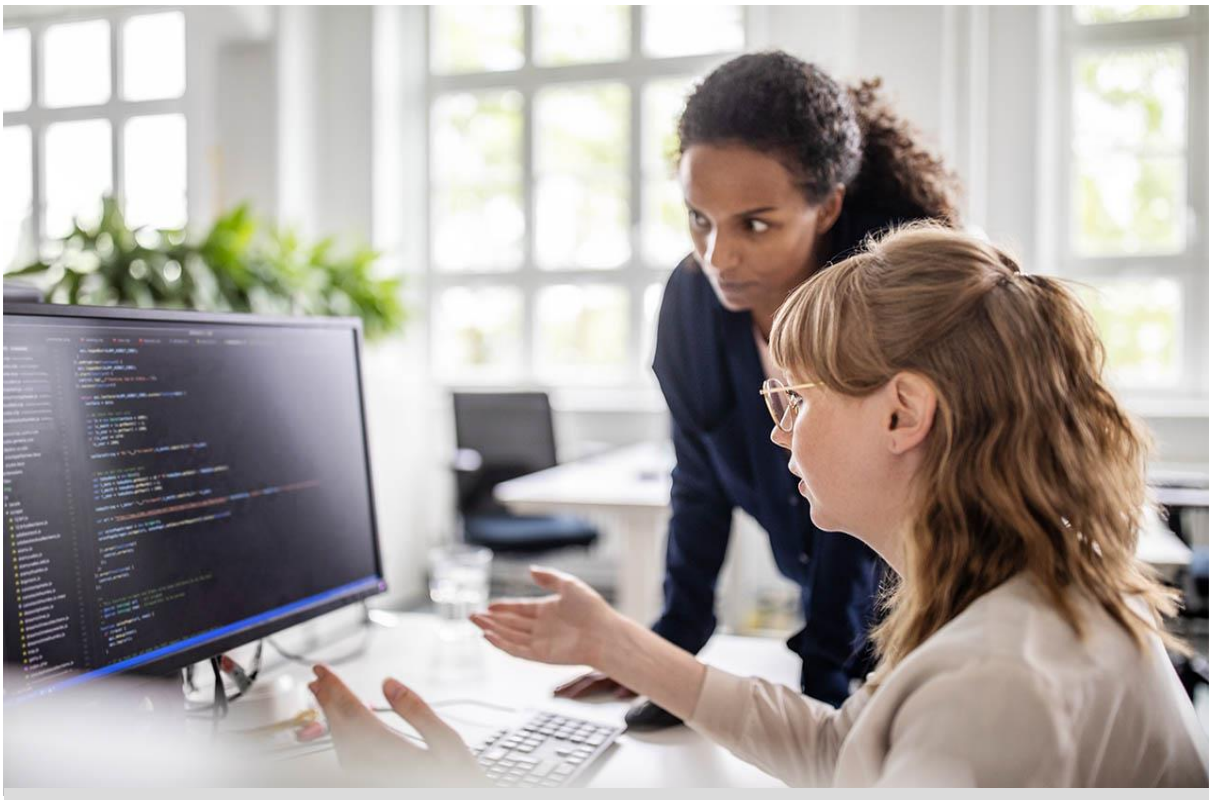


Javanista to Java Master

SKILLSOFT ASPIRE JOURNEY

skillsoft | percipio

Aspire Journeys

Javanista to Java Master

16

skillsoft

Earn a Badge

Javanista to Java Master is the journey intended for Java developers who aspire to sharpen their skills in Java enterprise frameworks and wish to become master's in enterprise application development.

In this journey, you will first learn how to write and run JUnit tests and identify the right situation to construct and use mock object. The journey continues by discovering how Jakarta Enterprise Edition (JEE) offers a straightforward and comprehensive approach to building state-of-the-art enterprise applications that enables you to create rich web-based interfaces using Java Server Faces (JSF) and effectively construct business logic using Enterprise JavaBeans (EJB). Finally, the journey takes you deeper into the Spring framework where you gradually progress from the Spring principles, Inversion of Control and Aspect Oriented Programming to the Database Integration using ORM frameworks such as JPA & Hibernate, Spring Data JDBC, and Spring Data JPA. You will complete this journey with a clear exploration on Spring Batch for processing batch data.

View Less

28 courses | 40h 15m 11s

Tracks

Track 1: Unit Testing

In this track of the Javanista to Java Master journey, the focus will be on how to write and run JUnit tests and identify the right situation to construct and use mock object.

Explore 8 courses | 10h 55m 49s

Track 2: Java EE

In this track of the Javanista to Java Master journey, the focus will be on how Jakarta Enterprise Edition (JEE) offers a straightforward and comprehensive approach to building state-of-the-art ente...

View More

Explore 5 courses | 7h 53m 20s

Track 3: Spring Basics

In this track of the Javanista to Java Master journey, the focus will be on Spring principles, Inversion of Control and Aspect Oriented Programming, and database Integration using ORM frameworks suc...

View More

Explore 7 courses | 11h 2m 46s

Track 4: Spring Data

In this track of the Javanista to Java Master journey, the focus will be on working with Spring Data JDBC, Spring Data JPA, and Spring Batch for processing batch data.

Explore 8 courses | 10h 23m 14s

Unit Testing: An Introduction to the JUnit Framework

Objectives:

- describe various aspects of software testing and test-driven development
- identify the significant features of the JUnit framework that simplify the writing of test cases
- create a Java project using an Apache Maven archetype that includes a separate test directory
- recognize what steps may be involved when testing an application without the JUnit framework
- describe the libraries and annotations that can be used to define JUnit tests
- run JUnit tests for your application, both collectively and individually
- structure your test executions by implementing a common set up and clean up method
- define a common set up and tear down method for all tests in a JUnit class

Unit Testing: Assertions & Assumptions in JUnit

Objectives:

- set a test to check that an actual value generated by a unit in an app matches the expected value
- implement test cases that verify Boolean data and objects
- develop test cases that ensure a unit of an app can run within a specified time
- build tests to verify that method calls throw an exception when invalid data is passed to it
- define a group of asserts and check whether two iterable objects contain the same data
- configure a test case to execute only when a specified assumption is satisfied
- set a block of code to execute when a specified assumption is met

Unit Testing: Advanced Annotations in JUnit

Objectives:

- use the DisplayName annotation to set a descriptive name for a test case
- apply JUnit annotations to set test cases to only run on specific operating systems
- apply JUnit annotations to set test cases to only execute on specific Java runtime environments
- enable or disable a test case based on system properties or environment variables
- create a customized JUnit annotation composed of several built-in annotations
- specify an order for test case executions within a test class
- set JUnit test cases to execute concurrently

Unit Testing: Parameterized JUnit Tests

Objectives:

- apply the RepeatedTest annotation to run a test case multiple times
- use the ValueSource annotation to run a parameterized test case several times for a set of pre-defined parameters
- set parameterized tests to run for null and empty parameter values as well as CSV values
- define a class to apply a transformation to parameters before they are fed into a test case

Unit Testing: Executing JUnit Tests

Objectives:

- compile and run enabled test cases in your project using Apache Maven
- use Apache Maven to execute only specific test cases based on class name, method name, or tags
- compile and run enabled test cases in your project using the JUnit ConsoleLauncher
- specify test cases to be included or excluded in a test execution launched from the ConsoleLauncher based on class name or tags

Unit Testing with Mocks: Getting Started with Mockito

Objectives:

- recall how mocks can be used with unit tests
- recall the benefits of using Mockito to mock objects
- install Maven and set up a new project
- set up IntelliJ with a Maven project
- add dependencies for JUnit and Mockito
- use the static mock() method to mock objects
- use the when().thenReturn() syntax to configure behavior
- use assertions in test cases
- configure mocks with the when().thenReturn() syntax with input arguments
- create mock objects of interfaces
- sort a treeset using a comparator mock
- set up a real-world object that can be mocked
- mock an iterator object

Unit Testing with Mocks: Creating Mocks & Verifying Behavior

Objectives:

- create mocks using the @Mock annotation
- set up multiple objects to be mocked in test cases
- mock objects in a real-world scenario
- use @ExtendWith to automatically initialize mocks
- illustrate how the @InjectMock annotation works
- use the @InjectMock annotation to inject mocks
- use the @InjectMocks annotation to inject mocks into constructors, setters, and fields
- verify the invocations of a method
- use the verifyNoInteractions() function
- illustrate the nuances of the verify() function
- demonstrate how ordered mocks work
- use multiple techniques to verify the behavior of objects under test and their interactions with mocks

Unit Testing with Mocks: Mocking Exceptions & Using Spies

Objectives:

- configure stubs to throw exceptions
- throw exceptions from stubbed methods
- throw exceptions from methods that return void
- create partial mocks using spies
- stub individual methods in spies
- use spies with interfaces
- verify interactions with spies
- use the doNothing() function with void methods
- work with argument captors
- track input arguments to void methods using argument captors

Building User Interfaces with JSF: Getting Started with Jakarta Server Faces

Objectives:

- give a theoretical and conceptual overview of Jakarta Server Faces
- set up Java and a build utility for JSF application development
- download and set up an Apache TomEE web server where a JSF app can be deployed
- define the dependencies and configurations for a simple JSF application
- develop the backing bean and a Facelet with the UI components for a JSF app
- package, deploy, and test a JSF application
- incorporate a CSS file and images in your JSF Facelets
- render data in a tabular form using a JSF dataTable
- recognize the HTML elements that correspond to commonly used JSF tags

Building User Interfaces with JSF: Building User Interactions with JSF

Objectives:

- configure page navigation using buttons in a JSF Facelet
- define navigations rules for a JSF web application in a configuration file
- develop and set up methods that are triggered once a user changes a value in an input element
- configure a JSF app to invoke a method once a user has interacted with a UI component
- create a validator method that will check for the correctness of user input
- configure user-defined and JSF built-in validators to check user input in a form
- recognize the scope of a bean set to last for the duration of a client session
- identify the components that enable form validation and the display of validation errors in an input form

Building User Interfaces with JSF: Integrating a Database with a JSF App

Objectives:

- create a table in a relational database and populate it with data
- set up a JSF application to use JDBC to connect to a database and run a SELECT query
- configure a Facelet to display the results of a SELECT query execution on a database
- allow users of a JSF app to input data to be inserted into a database
- deploy and test a JSF app that enables insert operations on a database
- define the methods that enable the editing and updating of existing data in a database
- configure the UI of your JSF app to enable editing and updating
- develop a method and configure the UI of your JSF app to allow existing data to be removed from a database

Encapsulating Business Logic with Jakarta: An Overview of Enterprise Beans

Objectives:

- outline the features, use cases, and categories of Jakarta Enterprise beans
- download and install the Wildfly (formerly Jboss) application server
- configure a Maven project that can be used to develop and deploy Jakarta Enterprise Beans
- implement a basic stateless bean that returns text when a method is invoked and then deploys it to an app server
- develop a client application to locate and connect to a remote session bean and use it to invoke methods
- execute a program that invokes a remote session bean
- create a stateless session bean that attempts to maintain some type of state
- recognize the limitations of a stateless session bean
- distinguish between stateless and stateful session beans
- develop, deploy, and access a singleton session bean
- apply degrees of concurrent access to the methods in a singleton bean

Encapsulating Business Logic with Jakarta: Advanced Topics in Enterprise Beans

Objectives:

- create and configure a messaging queue on a Wildfly application server instance
- define a message-driven bean that consumes text messages placed on a message queue
- develop an application that connects to a message queue and places text messages on that queue
- using the enterprise bean timer service to set a method to execute after a specified delay
- create a client application that invokes a task configured with the enterprise bean timer service
- set a task to execute multiple times using a ScheduleExpression
- configure a datasource in Wildfly that links up with a relational database
- create an entity class in an application that maps to a table in a relational database
- use an EntityManager instance to query data from and persist data to a database from a session bean
- develop an application that transmits objects to a bean for persistent storage in a database

Spring: An Overview of the Spring Framework

Objectives:

- describe Spring framework and recognize how it can help Java developers
- recall how dependency injection and inversion of control are related and how these relate to beans in Spring
- recognize how Spring Boot can be used to simplify application development
- delineate a number of different features in an application whose development can be simplified using various modules in the Spring framework
- outline the model-view-controller design paradigm and how Spring can help implement it for an application
- recognize the features and use cases of Spring data and recall how it eases data integration

Spring Framework: The Fundamentals of Dependency Injection

Objectives:

- recognize the concept of dependency injection and inversion of control
- describe the features available in the Spring framework to implement dependency injection
- define a bean to be injected into an application using an XML file
- use the Spring framework's BeanFactory to instantiate a bean and inject it into a program
- create an ApplicationContext instance to provision and inject beans
- contrast the initialization of beans in an ApplicationContext with their creation in a BeanFactory
- define a bean using annotations in a Java config file
- set methods to run after bean construction and before their destruction
- set the properties of a bean using methods in a Java config file
- set the properties of a bean using tags in an XML file

Spring Framework: Configuring Beans with Autowiring

Objectives:

- link a bean to a dependent bean using a reference set in the XML config file
- recognize the different ways in which bean references can be set
- use autowiring to connect beans that are related to each other
- apply autowiring to the beans created from a class using the Autowired annotation
- apply the Autowired annotation to the setters and constructors of a bean class
- outline the limitations of autowiring
- implement autowiring using a Java-based bean configuration file
- create a class whose beans depend on several other beans
- set autowiring for a bean with multiple dependencies using an XML config file
- set autowiring for a bean with multiple dependencies using a Java config file

Spring: Extending Applications with Aspect-oriented Programming

Objectives:

- describe what aspect-oriented programming is and when it is applicable
- recognize what mechanisms are available to implement aspect-oriented programming in Java
- create and configure a project where aspect-oriented programming can be applied
- define a method that can be set as an aspect in a Java application
- execute a program with aspects and recognize how such aspects and pointcuts are related
- recognize different ways to define pointcuts
- set up aspects in an application using Java annotations
- define pointcuts in your application that apply to methods in multiple classes
- create pointcuts that have overlapping join points
- configure a pointcut using the within designator in order to map to join points within a class or package
- define pointcuts that are composed of other pointcuts
- create a custom annotation in order to explicitly define the methods that must match a pointcut

Spring: Exploring Advices in Aspect-oriented Programming

Objectives:

- create a project where aspects can be configured with an XML file
- set aspects to run before and after certain methods in your application using before and after advices
- describe the behavior of and recognize the use cases of the around advice in aspect-oriented programming
- execute aspects after a method has executed successfully or when it throws an exception
- implement different types of advices in your aspect-oriented application using Java annotations
- identify the order in which advices will be applied to join point methods
- identify the use cases of aspect-oriented programming and implement a logging feature in your application using this programing paradigm

Database Integration: Overview of Using JPA & Hibernate

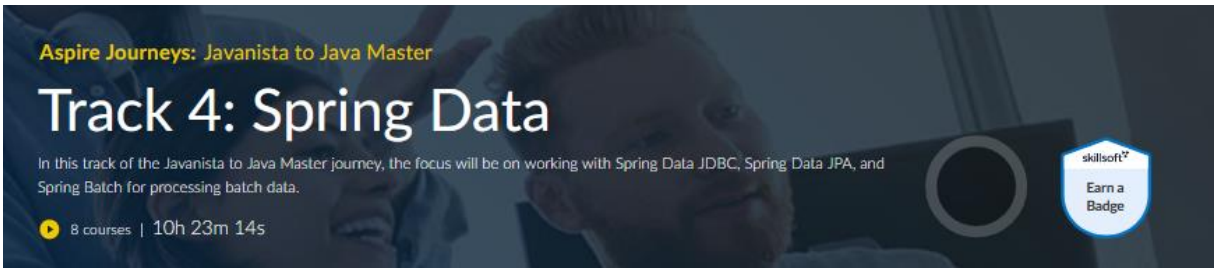
Objectives:

- describe object-relational mapping and the role of JPA and Hibernate in implementing this technique
- use Apache Maven to create a Java project that uses the Hibernate framework
- integrate an application with a relational database using Hibernate configurations and APIs
- apply Hibernate APIs to persist the data in Java objects to a relational database
- use JPA configurations and annotations to define the entities in a Java app and map them to a relational database
- define how a database schema will be set up once a JPA app is launched
- use SQL scripts to set up a database for use by a Java app

Database Integration: Modeling Data Using JPA & Hibernate

Objectives:

- configure entities in an application using JPA annotations
- set identifiers for entities in an application to auto-generate
- define a class whose members will serve as the composite key for an entity in an application
- recognize different ways to define composite keys for the entities in an application
- use the JPA APIs to add new data to and retrieve existing data from a table in a relational database
- perform update and delete operations on persisted data using JPA APIs
- define entities in an application that have a one-to-one relationship with one another
- describe how Hibernate translates a one-to-one relationship defined with JPA annotations to a relational database
- implement a one-to-many relationship between entities in an application using JPA
- use JPA annotations to configure a many-to-many relationship between entities in an application



Working with Spring Data JDBC: An Introduction to JDBC & Spring

Objectives:

- recognize the features of JDBC and how it simplifies database access
- identify the features of Spring, Spring Data, and Spring Data JDBC that can be used to implement database operations from a Java application
- use Spring Initializr to generate a Spring Boot application that uses Spring Data JDBC
- use a JdbcTemplate instance to connect to a database and run a query
- execute create, read/retrieve, update, and delete queries against a database using a JdbcTemplate instance
- define an entity class that maps to a relational database table
- apply Spring's RowMapper to operate on each row of a query execution's results and generate a Java object
- implement read and update operations on a database with a data access layer class

Working with Spring Data JDBC: Spring Data JDBC & the CrudRepository

Objectives:

- map an entity in your application to Spring Data's CrudRepository and use it to persist Java objects to a database
- map custom queries to a method in a CrudRepository
- define database tables and their corresponding entities in a Spring application to represent a many-to-one relationship
- create and persist objects that are related to one another in a many-to-one form
- configure database tables and their corresponding entities in a Spring application to represent a many-to-many relationship
- instantiate and persist objects that have a many-to-many relationship with one another

Working with Spring Data JDBC: Configuring Derived Query Methods

Objectives:

- enable logging in your Spring application to view the translation of CrudRepository methods into SQL queries
- call various CrudRepository methods that are implemented by default by the Spring framework
- configure derived query methods that search for entities based on a parameter
- set up derived query methods that search for entities based on the values of multiple properties
- configure derived query methods that search the numeric fields of entities for values in a given range
- set up derived query methods that return entities in a specified order

Working with Spring Data JPA: The Fundamentals of Spring Data JPA

Objectives:

- outline the fundamentals of object-relational mappers and JPA
- identify Spring framework projects and where Spring Data JPA fits in
- create a project that uses Spring Data JPA, Hibernate, and a relational database
- set up a class that represents an entity to be persisted into a database table using Spring Data JPA
- define a DataSource and related beans in your application to enable data persistence using Spring Data JPA
- configure a CrudRepository that enables basic CRUD operations on data
- write the contents of Java objects into a database table
- use Spring Boot and Spring Initializr to set up a project that adopts Spring Data JPA
- load entity data from Java objects to a database table from a Spring Boot app
- define a DataSource to be used in a java application within a properties file

Working with Spring Data JPA: Derived Query Methods

Objectives:

- retrieve and read entities from a database table using a CrudRepository's findBy methods
- recognize the exact syntax required to configure findBy methods in a CrudRepository
- set up update and delete operations in a CrudRepository
- implement derived query methods that operate on multiple parameters
- identify the different ways to implement exact match lookups using derived query methods
- search for entities based on patterns in a string property
- search for entities based on ranges of values in numeric and date fields

Working with Spring Data JPA: Custom Queries

Objectives:

- apply the @Query annotation to map queries to be run against a database to a method in a CrudRepository
- define custom queries that bind to parameters passed to a CrudRepository method
- configure update and delete operations in a CrudRepository
- define custom queries that can be referenced by names in a CrudRepository
- define named queries in an XML file and a Java source file

Processing Batch Data: The Fundamentals of Spring Batch

Objectives:

- describe how batch processing works and recognize its common limitations
- recognize the features offered by Spring Batch to make batch processing easier and more robust
- set up a Java project using Apache Maven in order to build a Spring Batch application
- set some of the core components of a Spring Batch application, such as a data source and transaction manager
- define an ItemReader component to read data from a CSV file and create Java objects for each row in the input file
- define an ItemProcessor to transform data and an ItemWriter to write data into an XML file
- create the class for the Java objects that will serve as the focal point in a batch process
- execute a Spring Batch process and identify the role played by various components in the application

Processing Batch Data: Spring Batch Configurations and Transformations

Objectives:

- define some of the core components of a Spring Batch application using Java annotations
- configure the ItemReader, ItemWriter, and ItemProcessor using Java annotations
- execute a Spring Batch process with annotation-based configurations and verify the results
- build a customized parser to translate a Java object into a string to be used in an XML file
- develop a customized FieldSetMapper to load CSV data into Java objects
- configure a batch process to transform XML input into CSV
- create a batch process to read data from a JSON file and load it into a relational database table
- read data from a database and write its contents into a JSON file

Business & Leadership for Javanista to Java Master

Optional

COURSE

Developing and Supporting an Agile Mindset

1169

COURSE

Encouraging Team Communication and...

1718

COURSE

The Essential Role of the Agile Product Owner

299

COURSE

Using Strategic Thinking to Consider the Big Picture

670

COURSE

Getting to the Root of a Problem

1016

COURSE

Unleashing Personal and Team Creativity

918

COURSE

Contributing as a Virtual Team Member

2854

COURSE

Developing a Growth Mindset

2633

COURSE

Developing a Successful Team

639

COURSE

Reaching Sound Conclusions

298

Bookshelf

Optional

BOOK

JUnit in Action, Third Edition

BOOK

Testing Java Microservices: Using Arquillian, Hoverfly,...

8

BOOK

Java Unit Testing with JUnit 5: Test Driven Developmen...

24

BOOK

Introducing Jakarta EE CDI: Contexts and Dependency...

BOOK

Beginning Jakarta EE: Enterprise Edition for Java:...

5

BOOK

Jakarta EE Recipes: A Problem-Solution Approach

BOOK

Full-Stack Web Development with Jakarta EE and Vue.js:...

1

BOOK

Pro Spring MVC with WebFlux: Web Developme...

4

BOOK

Spring Cloud Data Flow: Native Cloud Orchestration...

2

BOOK

Spring Security in Action

1

BOOK

Learn Microservices with Spring Boot: A Practical...

7

BOOK

Spring Quick Reference Guide: A Pocket Handbook...

9

BOOK

Spring Boot Persistence Best Practices: Optimize Java...

10

BOOK

Pro Spring Security: Securing Spring Framework 5 and...

4

BOOK

Spring Boot and Single-Page Applications: Securing Your...

5

BOOK

Spring Boot 2: How to Get Started and Build a...

7

BOOK

Beginning Spring 5: From Novice to Professional

6

BOOK

Spring Boot in Action

2

BOOK

Spring in Action, Fifth Edition

38

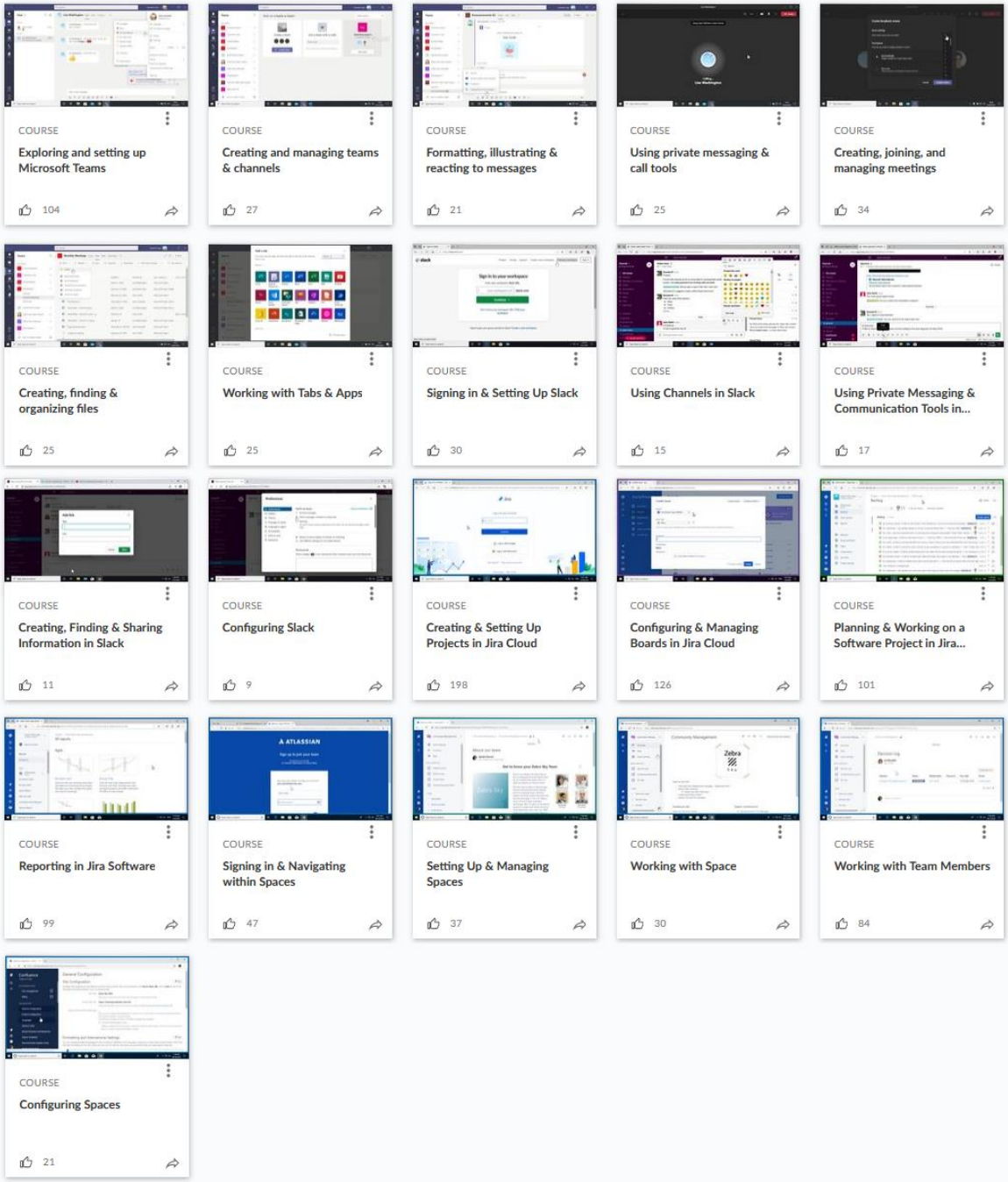
BOOK

Java APIs, Extensions and Libraries: With JavaFX,...

4

Productivity Tools for Javanista to Java Master

Optional



FOLLOW US ON:

www.skilltech.pl
email: biuro@skilltech.pl
tel. +48 22 44 88 827

SkillTech
Technology hired for excellence