Developer to Software Architect

SKILLSOFT ASPIRE JOURNEY

skillsoft

Głównym wyzwaniem przed którym stają dziś organizacje na całym świecie jest konieczność ciągłego podnoszenia umiejętności i poziomu wiedzy w ślad za gwałtownym rozwojem nowych technologii i zmian na globalnym rynku.

Stały rozwój i podnoszenie kwalifikacji w IT od dawna jest już rzeczą oczywistą, a możliwość zapewnienia wsparcia specjalistom chcącym stale się rozwijać jest jedną z głównych kart przetargowych w walce o pracownika.

Na rynku liczą się dziś ludzie, którzy posiadają konkretne kompetencje i zestaw umiejętności pozwalający im wykonywać zadania efektywnie, a nie Ci z najdłuższym stażem pracy.

Dziś, bardziej niż kiedykolwiek w cenie jest umiejętność budowania ścieżki kariery dla profesjonalistów IT, którzy wciąż chcą się liczyć na rynku pracy.

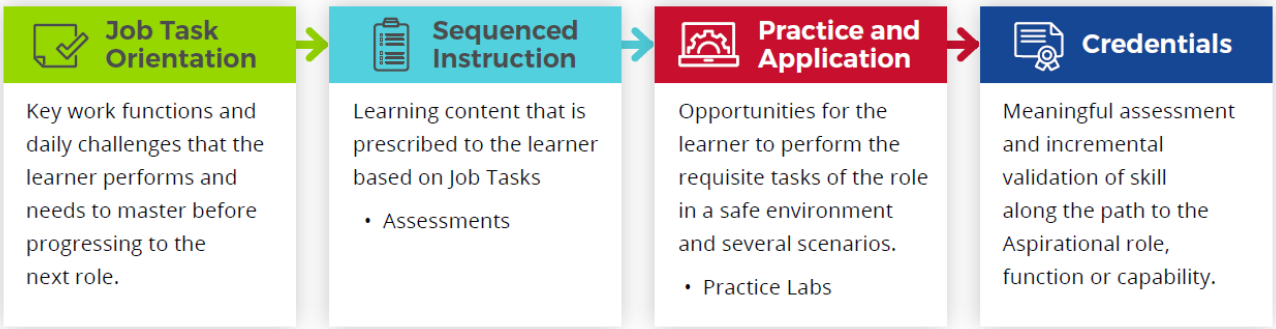
Skillsoft Aspire Journey stanowi odpowiedź na pytanie, jakie szkolenia muszę ukończyć, aby być przygotowanym do mojej wymarzonej pracy. Spośród kilkuset kanałów tematycznych dostępnych na naszej platformie szkoleniowej nasi specjaliści wybrali te, które naszym zdaniem najlepiej wyposażą uczących się w narzędzia potrzebne do realizacji zadań w nowej roli.

Skillsoft Aspire Journey to zestawy szkoleń i ćwiczeń w języku angielskim, które metodycznie, krok po kroku pozwalają specjalistom przejść od poziomu podstawowego do zaawansowanego.

Każda ścieżka zawiera szkolenia, laboratoria wirtualne, video i książki, które pomogą uczącym się osiągnąć pożądane kompetencje poświadczone certyfikatem.

Aspire Journey Model

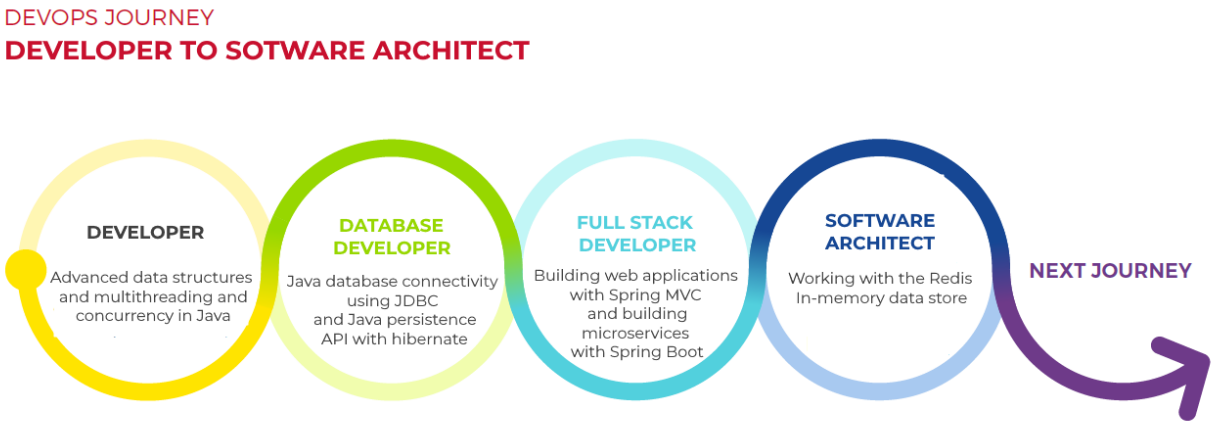
Cała ścieżka opiera się na 4-elementowym cyklu powtarzanym na kolejnych etapach nauki.



1. Określenie kluczowych funkcji i wyzwań, z którymi musi poradzić sobie uczący się w chwili obecnej, jak i tymi, z którymi przyjdzie mu się zmierzyć w nowej pracy.
2. Przejście zaprojektowanych ścieżek w proponowanej kolejności, wykonanie ćwiczeń i zaliczenie testów.
3. Przećwiczenie nowych umiejętności w kontrolowanym środowisku w oparciu o gotowe scenariusze działań. Laboratoria wirtualne Skillsoft
4. Certyfikat – zaliczenie testu końcowego na poziomie co najmniej 70% i uzyskanie certyfikatu potwierdzającego ukończenie danego etapu nauki.

Aspire Journey – Developer to Software Architect

Analizując trendy opisujące zachowanie użytkowników na naszych platformach szkoleniowych i współpracując ściśle z naszymi klientami na całym świecie Skillsoft wyselekcjonował najlepsze materiały szkoleniowe i ułożył je w ustrukturalizowaną ścieżkę rozwoju. Ścieżka zawiera niemal 53 godziny szkoleniowe.



13 courses
19h 41m 28s

- advanced data structures and multithreading
- concurrency in Java

10 courses
16h 13m 55s

- Java database connectivity using JDBC
- Java persistence API with hibernat

9 courses
11h 41m 03s

- building web applications with Spring MVC
- building microservices with Spring Boot

6 courses
5h 14m 20s

- working with the Redis In-memory data store

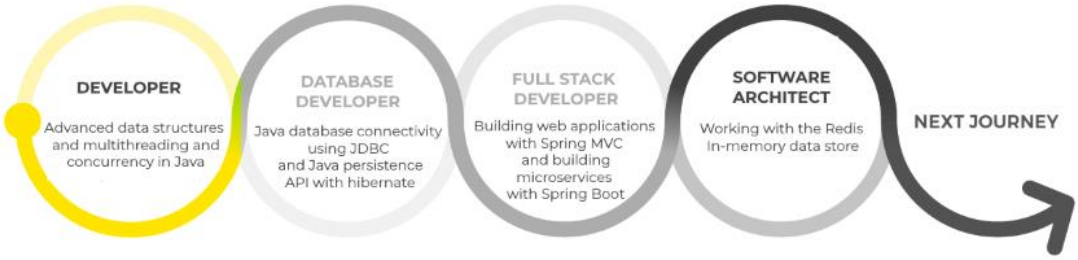
PREREQUISITES

In order to fully profit from the potential of this Aspire Journey, you should:

- be familiar with Java
- be familiar with database development

DEVOPS JOURNEY

DEVELOPER TO SOTWARE ARCHITECT



Track 1: Developer (duration: 19h 41m 28s)

 Advanced Data Structures & Algorithms in Java: Working With Binary Trees	 Advanced Data Structures & Algorithms in Java: Solving Binary Tree Problems
Objectives ▪ recognize the characteristics of binary trees ▪ summarize breadth-first traversal in a binary tree ▪ summarize depth-first traversal in a binary tree ▪ explain depth-first, pre-order traversal in a binary tree ▪ explain depth-first, in-order traversal in a binary tree ▪ explain depth-first, post-order traversal in a binary tree ▪ represent a binary tree using the Java programming language ▪ write and execute code which performs breadth-first traversal ▪ visualize the levels of nodes visited in breadth-first traversal ▪ use in-order traversal to traverse nodes via the left child, parent, and then right child ▪ implement in-order traversal using recursion ▪ use pre-order traversal to traverse nodes via the parent, left child, and then right child ▪ implement pre-order traversal using recursion ▪ use post-order traversal to traverse nodes via the left child, right child, and then parent ▪ implement post-order traversal using recursion	Objectives ▪ recognize how the recursive algorithm to count nodes works ▪ write code to implement the count nodes algorithm ▪ recognize how the maximum depth of a binary tree is calculated ▪ calculate the maximum depth of a binary tree ▪ explain how the sum of nodes along a path can be computed ▪ compute and check to see whether a path with a certain sum exists ▪ describe the original binary tree and its mirror ▪ implement the mirroring of a binary tree ▪ recognize a full binary tree ▪ write code to check whether a binary tree is full ▪ recognize a perfect binary tree ▪ write code to check whether a binary tree is perfect ▪ recognize a complete binary tree ▪ write code to check whether a binary tree is complete ▪ recognize a balanced binary tree ▪ write code to check whether a binary tree is balanced

<table>
<tr>
<td> 

Advanced Data Structures & Algorithms in Java: Working with Binary Search Trees</td>
<td> 

Advanced Data Structures & Algorithms in Java: Sorting & Searching Algorithms</td>
</tr>
<tr>
<td>Objectives

▪ describe the constraints on the nodes of a binary search tree
▪ explain how insertion works in a binary search tree
▪ describe how lookup works in a binary search tree
▪ insert a node into a binary search tree
▪ look up a node in a binary search tree
▪ find the minimum and maximum value in a binary search tree
▪ print a range of values in a binary search tree
▪ see if a binary tree meets the constraints of a binary search tree</td>
<td>Objectives

▪ recognize the trade-offs involved in choosing sorting algorithms
▪ describe the selection sort algorithm
▪ write code to implement selection sort
▪ explain the bubble sort algorithm
▪ write code to implement bubble sort
▪ implement adaptive bubble sort with early stopping
▪ define the insertion sort algorithm
▪ write code to implement insertion sort
▪ summarize the shell sort algorithm
▪ write code to implement shell sort
▪ describe the merge sort algorithm
▪ develop methods for splitting and merging operations
▪ write code to implement merge sort
▪ define the quick sort algorithm
▪ write code to implement quick sort
▪ describe the binary search algorithm
▪ design and write code for linear search
▪ design and write code for binary search
▪ design and write code for jump search
▪ design and write code for interpolation search</td>
</tr>
</table>

<table>
<tr>
<td> 
</td>
<td>Advanced Data Structures & Algorithms in Java: Working with the Binary Heap</td>
<td> 
</td>
<td>Advanced Data Structures & Algorithms in Java: Getting Started with Graphs</td>
</tr>
<tr>
<td colspan="2">Objectives

▪ explain the common operations to be performed on a priority queue
▪ describe how a binary heap data structure functions
▪ explain the array implementation of a binary heap
▪ summarize the heapify operation on a binary heap
▪ recognize how insert and remove works on a binary heap
▪ write code for the base class operations in a binary heap
▪ implement the maximum heap
▪ perform operations on a maximum heap
▪ implement the minimum heap
▪ recognize the functioning of the heap sort algorithm
▪ describe how the heapify in the heap sort works
▪ explain how the heap sort is used for sorting
▪ write code for the heapify operation in heap sort
▪ implement the sort functionality in heap sort</td>
<td colspan="2">Objectives

▪ define the structure and components of a graph
▪ recognize directed and undirected graphs and their use cases
▪ recognize the similarities between graphs and trees
▪ explain weighted graphs and their use cases
▪ describe how a graph can be represented as an adjacency matrix
▪ define the adjacency list representation of a graph
▪ contrast the adjacency set representation with the adjacency list
▪ write code to represent graphs using an adjacency matrix
▪ perform operations on an adjacency matrix graph
▪ represent directed and undirected graphs using an adjacency matrix
▪ represent weighted graphs using an adjacency matrix
▪ write code to represent graphs using an adjacency list
▪ restructure code to allow weighted graph representations
▪ write code to represent graphs using an adjacency set
▪ understand how graph traversal techniques work
▪ perform breadth-first traversal on a graph
▪ perform depth-first traversal on a graph</td>
</tr>
</table>

<table>
<tr>
<td> 
</td>
<td>Advanced Data Structures & Algorithms in Java: Working with Graph Algorithms</td>
<td> 
</td>
<td>Multithreading and Concurrency in Java: Introduction to Concurrent Programming</td>
</tr>
<tr>
<td colspan="2">Objectives

▪ explain what is meant by the topological sorting of graph nodes
▪ describe the topological sorting algorithm
▪ write code to implement topological sort
▪ design a graph representation of courses and perform topological sort
▪ explain the shortest path algorithm for unweighted graphs
▪ populate the distance table and perform backtracking
▪ write code to implement the shortest path algorithm
▪ contrast the algorithms for shortest path in unweighted and weighted graphs
▪ identify when the shortest path to a node needs to be updated
▪ describe Dijkstra's algorithm for the shortest path in a weighted graph
▪ write code to implement Dijkstra's algorithm</td>
<td colspan="2">Objectives

▪ identify a sequential execution plan as well as its limitations
▪ describe a multithreaded approach to tackling multiple tasks
▪ distinguish between multithreading and multi-processing and describe the benefits and drawbacks of each approach
▪ recognize a race condition and how this could lead to inconsistent outcomes
▪ describe how synchronization of concurrent threads can be achieved using locks
▪ identify use cases for semaphores and describe how they can be used
▪ recognize the conditions under which concurrent threads may become deadlocked and how this can be prevented
▪ recall Java objects that can be used to enable and simplify working with a multithreaded application
▪ compare and contrast synchronization using locks with the use of atomic operations on atomic variables
▪ describe the benefits of thread pools when it comes to management of multiple concurrently executing threads
▪ list use cases of the Future object in Java and the Fork/Join framework when it comes to executing multiple threads</td>
</tr>
</table>

Multithreading and Concurrency in Java: Spawning & Launching Threads

Objectives

- define a task to be executed within a Java thread by implementing the Runnable interface
- extend the Thread class to define a task to be run in a thread
- recognize the effect of the start() method in kicking off Java threads
- use the join() method to synchronize one thread with the execution of another
- identify the different states of a thread and the possible transitions from them
- verify whether a thread is active by invoking the isAlive() method
- recognize the possible and default values for a Java thread's priority
- distinguish between user and daemon threads in Java
- build an application that implements multithreading to download multiple web pages concurrently
- identify situations where multithreading can be applied to speed up executions
- describe the mechanism of thread interruption in Java

Multithreading and Concurrency in Java: Thread Synchronization & Locks

Objectives

- recognize the situations where a race condition could occur in Java
- implement synchronized functions to manage multiple threads updating the same resource
- differentiate between synchronized functions and synchronized blocks in Java
- recall the conditions under which concurrent threads could encounter a deadlock in Java
- identify some of the options available to prevent the occurrence of deadlocks
- introduce a ReentrantLock in your program to enable exclusive access to a shared resource
- use the tryLock() method of a ReentrantLock to prevent a potentially long wait for a resource
- configure a thread to wait on a resource for a limited amount of time by using the tryLock() method of a ReentrantLock
- implement a StampedLock to regulate access to a shared resource
- distinguish between a readLock and writeLock for a Java StampedLock

Multithreading and Concurrency in Java: Concurrency & the Producer-Consumer Problem

Objectives

- define a producer task that can run concurrently with a consumer using a shared queue
- develop a consumer task that works with a concurrent producer through a shared buffer
- recognize the behavior of producer and consumer threads when working with a shared bounded buffer
- use a ReentrantLock to access the shared queue for the Producer-Consumer problem
- recognize the role of conditions in enabling different threads to wait on different conditions on the same object
- verify the validity of a solution to the Producer-Consumer problem with multiple producer and consumer threads
- implement a solution to the Producer-Consumer problem using Java's built-in ArrayBlockingQueue

Multithreading and Concurrency in Java: Objects for Concurrent Programming

- Objectives
-
- implement semaphores in a multithreaded Java application where each thread is treated equally
- use semaphores in a multithreaded Java application where different threads require varying levels of permits to access a shared resource
- recognize the effects of updates to shared variables in a multithreaded Java applications
- recall the effect of declaring a shared variable to be volatile
- enable atomic operations on integer variables using an AtomicInteger object
- identify some of the different types of synchronized collection objects available in Java
- use a CopyOnWriteArrayList to synchronize updates from multiple threads
- recognize the effect of iterating over a collection while a write is being performed
- use a ConcurrentHashMap in a multithreaded Java application
- compare the performance of insert operations on various list and map data structures in Java

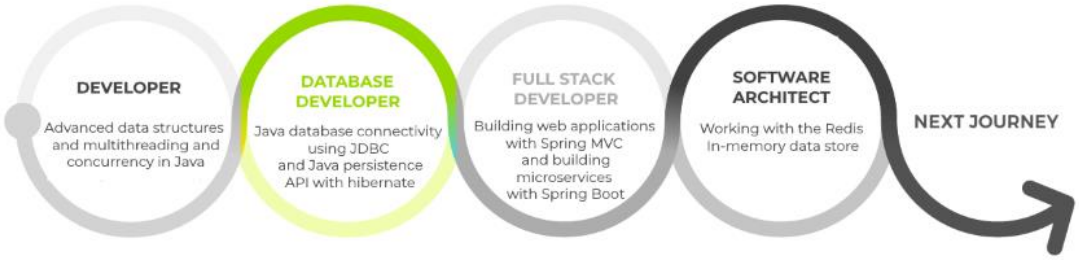
Multithreading and Concurrency in Java: Scaling a Multithreaded Application

Objectives

- create an ExecutorService instance to manage a pool of thread workers and submit tasks to it
- use a CountDownLatch to synchronize with the execution of threads submitted to an ExecutorService
- describe how an ExecutorService instance can be shut down while still allowing submitted tasks to run
- distinguish between the graceful and immediate shutdown of an ExecutorService instance
- recognize the options available to monitor the execution of a task using a Future object
- implement the Callable interface to define a task that can run in a thread and returns a result
- schedule one-off tasks to execute at a specified point in the future using the ScheduledExecutorService
- schedule recurring tasks to execute at a set interval using the ScheduledExecutorService
- use the Fork/Join framework to break up a large task into a number of smaller sub-tasks
- submit a large task to ForkJoinPool and recognize how these can be split into smaller sub-tasks

DEVOPS JOURNEY

DEVELOPER TO SOTWARE ARCHITECT



Track 2: Database Developer (duration: 16h 13m 55s)

 JDBC: An Introduction to Java Database Connectivity	 JDBC: Executing Statements
Objectives ▪ describe the role of JDBC and JDBC drivers in enabling interactions between a Java app and a database ▪ identify the different JDBC objects that enable a Java app to work with a database, and recall the roll played by each of them ▪ recognize the role played by a ResultSet in navigating over and modifying the contents of tabular data ▪ recall the specific types of RowSets available in JDBC and their specific features and use cases ▪ describe the role of batch updates and transactions when working with JDBC ▪ download and install MySQL and the MySQL Workbench tool ▪ write a Java app that connects to a database using JDBC ▪ configure a Java app to use a JDBC driver when connecting to a database ▪ set up a Maven project to work with a JDBC driver	Objectives ▪ identify the different ways to specify connection details when connecting to a database using JDBC ▪ use database connection details in a properties file to set up a connection to a database using JDBC ▪ adopt a DataSource object to create a Connection with a database ▪ execute a SQL query to create a database table from a Java app ▪ run a SELECT query against a database and access the returned tabular data in a JDBC ResultSet object ▪ add rows to a database table by running an INSERT query from a Java app ▪ modify the contents of a database table from a Java app by executing UPDATE and DELETE queries ▪ drop a table and database from a Java app ▪ recognize the limitations of the Statement object when several executions of similar queries are involved ▪ optimize the execution of similar queries by parametrizing them with PreparedStatements ▪ recall the types of exceptions that could occur when passing incorrect parameters to a PreparedStatement ▪ identify some of the exceptions that could be thrown when using JDBC to work with databases

JDBC: Working with ResultSets & Query Batches

Objectives

- identify the types of ResultSets available in JDBC
- scroll through a ResultSet to access data in various locations
- recognize the limitations of a TYPE_FORWARD_ONLY ResultSet
- refresh the contents of a ResultSet while iterating over its contents
- recognize the types of modifications that a ResultSet may not pick up
- modify the contents of a SQL table by updating a ResultSet
- remove data from a SQL table by performing deletes from a ResultSet
- identify potential issues when trying to modify a read-only ResultSet
- group similar INSERT queries together into a bath to optimize their executions
- combine the optimizations of batch executions and batch updates
- describe how the auto-commit feature may prevent queries from running as a transaction
- use the rollback feature of JDBC to undo modifications after an Exception is thrown
- perform a partial rollback by implementing Savepoints in a Java app

JDBC: RowSet

Objectives

- compare and contrast a JdbcRowSet and ResultSet
- perform scrolling and navigation over the contents of a JdbcRowSet
- update the contents of a SQL table through a JdbcRowSet
- use a CachedRowSet to work with a data while disconnected from the database
- recognize the limitations of working with a disconnected RowSet such as a CachedRowSet
- manipulate the contents of a CachedRowSet and push the updates to the underlying database table
- perform an offline join of two tables using a JoinRowSet
- update the contents of a SQL table through a JoinRowSet
- define a Predicate class that sets up a filter for a FilteredRowSet
- apply a Predicate instance to a FilteredRowSet to get it to include only data that fulfils the condition defined in the Predicate
- recognize the types of updates that can be performed on a FilteredRowSet
- describe the features offered by a WebRowSet
- recall how a WebRowSet records modifications to its contents

JDBC: Advanced Topics

Objectives

- load large image files into a SQL table using JDBC
- download image files from a SQL table into your file system
- use JDBC to upload and download text data to and from a database
- create and execute stored procedures from a Java application
- execute stored procedures that return values and retrieve the outputs generated
- use JDBC to run stored procedures that use the same parameter to process inputs and outputs

Java Persistence API: Getting Started With JPA & Hibernate

Objectives

- identify why object-relational mapping is needed to work with databases
- outline the basic functionality that the Hibernate framework provides and how it differs to JPA
- outline the basic functionality that the JPA framework provides and how it differs to Hibernate
- install MySQL and MySQL Workbench on a Windows machine
- set up an Apache Maven project on a Windows machine
- install MySQL and MySQL Workbench on a macOS machine
- set up an Apache Maven project on a macOS machine
- configure JPA and Hibernate dependencies in pom.xml
- represent entities and primary keys using annotations
- configure database connection details in persistence.xml
- store entities in the underlying database using the entity manager
- configure the range of actions that can be performed using persistence.xml
- configure drop and create actions using scripts

Java Persistence API: Configuring Fields & Performing CRUD Operations

Objectives

- recognize the primary key as a required field
- configure the table name using the @Table annotation
- use @GeneratedValue to generate primary key values
- contrast the AUTO and IDENTITY genaration types
- use the IDENTITY generation type for multiple tables
- use the SEQUENCE generation type
- use the TABLE generation type
- configure composite keys using @Embeddable and @Id annotations
- configure composite keys using @EmbeddedId annotation
- configure composite keys using the @IdClass annotation
- apply the @Column annotation to define columns
- specify precision and scale values for floating point columns
- specify not null and uniqueness constraints
- annotate non-persistable fields using @Transient
- use the @Temporal annotation for date fields
- use the @Lob annotation for large objects
- configure embeddable entities as persistent fields
- share embeddable objects across entities
- use begin and commit transactions
- perform read operations on database entities
- perform update and delete operations on entities
- specify entity attributes using XML

Java Persistence API: Mapping & Configuring Relationships

Objectives

- set up a one-to-one mapping between entities
- configure join columns for one-to-one mapping
- set up a bidirectional one-to-one mapping between entities
- configure one-to-one-mapping with a shared primary key
- configure one-to-one mapping with a join table
- set up a one-to-many unidirectional mapping
- configure a one-to-many mapping with a join table
- configure eager loading of entities on the many side
- configure lazy loading of entities on the many side
- configure one-to-many mapping with join columns
- retrieve many entities in order of specific attributes
- persist entities in a certain order
- set up a many-to-one unidirectional mapping
- configure a many-to-one mapping with multiple join columns
- configure a many-to-one mapping with a join table
- set up a one-to-many, many-to-one bidirectional mapping
- retrieve entities mapped in both directions
- configure the owning side and owned side in mappings
- set up a many-to-many bidirectional mapping

Java Persistence API: Embedding Collections & Managing Cascade Operations

Objectives

- define collection mapping for primitive data types
- configure lazy and eager loading for collections
- persist collections using the Set interface
- persist nested objects in collections
- persist collections defined using the Map interface
- configure maps using @MapKeyColumn and @MapColumn
- use embeddable objects as map values
- use entities as map values
- use entities as map keys
- recall the implications of cascade type NONE
- summarize the effects of cascade type PERSIST and REMOVE
- summarize the effects of cascade type MERGE
- summarize the effects of cascade type DETACH and ALL
- apply the @MappedSuperclass annotation to parent classes
- recall the effects of persisting derived entities with a mapped superclass
- use the SINGLE_TABLE inheritance to store all entities in a single table
- specify discriminator columns with specific discriminator values
- configure entities to handle null and unknown discriminator values
- use the TABLE_PER_CLASS inheritance to store entities in multiple tables
- use the JOINED inheritance to store entities in multiple tables

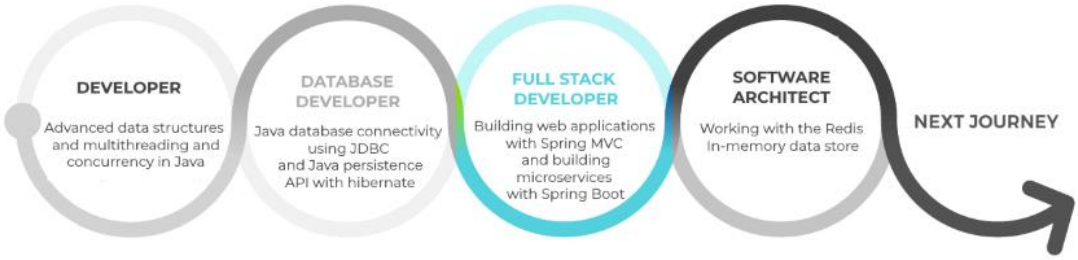
Java Persistence API: Executing Queries & Managing the Entity Lifecycle

Objectives

- execute native queries
- execute JPQL queries
- configure dynamic JPQL queries using named and positional parameters
- reference foreign keys within JPQL queries
- perform join operations using JPQL queries
- retrieve multiple fields using JPQL queries
- construct special objects to hold multiple fields within JPQL queries
- execute update and delete JPQL queries
- define and use named queries
- summarize the basic structure of the Criteria API
- run basic criteria queries
- run advanced criteria queries
- hook into the entity lifecycle before and after entity persistence
- hook into the entity lifecycle after load
- hook into the entity lifecycle before and after entity load
- hook into the entity lifecycle before and after entity removal
- configure an entity listener object for callbacks

DEVOPS JOURNEY

DEVELOPER TO SOTWARE ARCHITECT



Track 3: Full Stack Developer (duration: 11h 41m 3s)

 The Spring Web MVC Framework: Getting Started	 The Spring Web MVC Framework: Handling Requests & Errors
Objectives ▪ list the key features of the Spring framework ▪ outline how inversion of control is implemented in Spring ▪ identify the different projects in the Spring framework ▪ identify the need for dependency management in projects ▪ specify the components and interaction in the MVC architecture ▪ recognize the core technologies used in Spring MVC ▪ describe the essential elements that make up the Spring MVC architecture ▪ set up a Maven web application project using Eclipse ▪ run a Maven web application on the Tomcat server ▪ configure the application context programmatically ▪ run a Spring MVC application using Eclipse ▪ configure the application context using XML ▪ configure Spring beans using annotations ▪ specify a view resolver for logical view names	Objectives ▪ set up Spring MVC controllers to handle requests ▪ deploy WAR files directly to Tomcat on a Windows machine ▪ deploy WAR files directly to Tomcat on a MacOS machine ▪ configure multiple pages in your app ▪ configure multiple controllers in your app ▪ design the 3 tiers of your application ▪ separate each tier into a different package ▪ extract dynamic elements of a path ▪ access request parameters from an HTTP request ▪ inject request parameters into a controller method ▪ configure default values and optional request parameters ▪ handle exceptions using XML configuration ▪ specify multiple error pages for exception handling ▪ handle exceptions using programmatic configuration

<table>
<tr>
<td> 
</td>
<td>The Spring Web MVC Framework: Working With Forms & Files</td>
<td> 
</td>
<td>The Spring Web MVC Framework: Building Web Applications & REST APIs</td>
</tr>
<tr>
<td colspan="2">Objectives

▪ use forms in your application
▪ validate forms using built-in validators
▪ upload files to your application
▪ upload files using the ServletContextAware controller
▪ upload multiple files to your application
▪ upload multiple files to your application using multiple selections
▪ download files from your application</td>
<td colspan="2">Objectives

▪ download and set up starter templates for your application
▪ Set up login and register forms
▪ install MySQL and MySQL Workbench on Windows
▪ install MySQL and MySQL Workbench on MacOS
▪ configure the data source connection to MySQL
▪ implement three tiers of your web application
▪ register a new user in the application
▪ implement custom form validators
▪ perform form validation using buit-in and custom validators
▪ implement the REST API for read operations
▪ implement the REST API for create operations
▪ implement the REST API for update operations
▪ implement the REST API for delete operations</td>
</tr>
</table>

Spring Boot Microservices: Getting Started

Objectives

- recall the key features of the Spring framework
- recall the key features of Spring Boot
- install Apache Maven on a Windows machine
- install Apache Maven on a macOS machine
- import a Maven project into Eclipse
- build a Spring Boot application
- deploy and run a web application on the embedded Tomcat server
- deploy and run a web application on the embedded Jetty server
- deploy WAR files on an external Tomcat server
- use Spring Initializr to set up a project structure
- localize messages in different languages
- use Spring Tools Suite to set up a project structure
- configure the server port for your web application
- configure user-defined properties in your application

Spring Boot Microservices: Asynchronous Methods, Schedulers, & Forms

Objectives

- implement asynchronous methods on background threads
- schedule tasks at a fixed rate and with a fixed delay
- schedule asynchronous tasks using multiple threads
- extract request parameters and path elements
- use forms to get user input
- validate forms using built-in validators

Spring Boot Microservices: Building RESTful API Services

Objectives

- install Advanced REST Client on your machine
- perform read operations using HTTP GET requests
- perform create operations using HTTP POST requests
- perform update operations using HTTP PUT requests
- perform delete operations using HTTP DELETE requests
- install MySQL and MySQL Workbench on a Windows machine
- install MySQL and MySQL Workbench on a macOS machine
- integrate your application with MySQL
- perform CRUD operations with a database
- create and list records with a web user interface
- update and delete records with a web user interface
- cache responses from services
- evict caches with stale data

Spring Boot Microservices: Advanced Microservices & Securing Web Applications

Objectives

- send an email using the JavaMailSender API
- use interceptors to process requests and responses
- set up the Zuul edge service to act as an API proxy
- route and filter requests using Zuul
- send messages and make phone calls using Twilio
- set up the default login page
- configure in-memory users with Spring Security
- configure user and admin login roles
- set up the user entity and repository
- set up services and controllers for your application
- specify security settings for your application

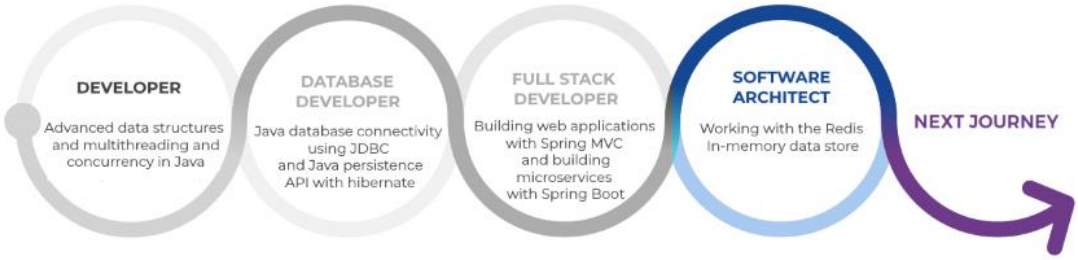
Final Exam: Full Stack Developer

Objectives

- access request parameters from an HTTP request
- build and use forms in a Spring Boot Web application
- build a Spring Boot application
- cache responses from services
- configure in-memory users with Spring Security
- configure multiple pages in your app
- configure the data source connection to MySQL
- configure the server port for your web application
- configure user and admin login roles
- deploy and run a web application on the embedded Jetty server
- deploy and run a web application on the embedded Tomcat server
- deploy WAR files directly to Tomcat on a MacOS machine
- deploy WAR files directly to Tomcat on a Windows machine
- describe the core technologies used in Spring MVC
- describe the different projects in the Spring framework
- describe the essential elements that make up the Spring MVC architecture
- download files from your application
- extract dynamic elements of a path
- identify the different projects in the Spring framework
- identify the need for dependency management in projects
- implement asynchronous methods on background threads
- implement form validation using built-in and custom validators
- implement in-memory users with Spring Security
- implement the REST API for create operations
- implement the REST API for update operations
- implement three tiers of your web application
- import a Maven project into Eclipse
- inject request parameters into a controller method
- install Advanced REST Client on your machine
- install Apache Maven on a MacOS machine
- install Apache Maven on a Windows machine
- install MySQL and MySQL Workbench on a macOS machine
- install MySQL and MySQL Workbench on a Windows machine
- install MySQL and MySQL Workbench on MacOS
- install MySQL and MySQL Workbench on Windows
- list the key features of the Spring framework
- perform create operations using HTTP POST requests
- perform delete operations using HTTP DELETE requests
- perform form validation using built-in and custom validators
- perform read operations using HTTP GET requests
- perform update operations using HTTP PUT requests
- recall the key features of the Spring framework
- recognize the core technologies used in Spring MVC
- register a new user in the application
- schedule tasks at a fixed rate and with a fixed delay
- send an email using the JavaMailSender API
- sending email using the JavaMailSender API
- separate each tier into a different package
- set up a Maven web application project using Eclipse
- set up Spring MVC controllers to handle requests
- set up the default login page
- specify multiple error pages for exception handling
- specify security settings for your application
- specify the components and interaction in the MVC architecture
- upload files to your application
- use built-in validators
- use forms in your application
- use Spring Tools Suite to set up a project structure
- validate forms using built-in validators
- validating forms using built-in validators

DEVOPS JOURNEY

DEVELOPER TO SOTWARE ARCHITECT



Track 4: Software Architect (duration: 5h 14m 20s)

 The Redis In-memory Data Store: An Introduction to Redis	 The Redis In-memory Data Store: Data Structures
Objectives ▪ describe the use cases of the Redis data store ▪ identify the types of data that can be stored in Redis and the properties of the different data structures ▪ recognize how Redis supports connections via various clients ▪ download and install Redis on your machine ▪ invoke create, read, update and delete operations from the Redis CLI ▪ create and update numeric data in Redis ▪ perform string operations on Redis data ▪ set and unset expiration for keys in a Redis store	Objectives ▪ recognize the features of list structures in Redis and illustrate the types of operations that can be performed with them ▪ extract elements from Redis lists individually or in bulk ▪ use the Redis hash structure to store and extract data in the form of a map ▪ distinguish between list and set data structures and apply the Redis CLI commands that apply to sets ▪ perform set operations such as union, intersection, and set difference on Redis sets ▪ describe the properties of sorted sets and illustrate how they contrast with regular Redis sets ▪ remove and modify the elements in a sorted set ▪ order elements with the same score in a sorted set based on their lexicographic values ▪ illustrate the use and limitations of the HyperLogLog structures when it comes to tracking unique values

The Redis In-memory Data Store: Messaging & Streaming

Objectives

- publish and subscribe to Pub/Sub channels using the Redis CLI
- set up a subscription to multiple Pub/Sub channels that match a pattern
- recognize the content and format of stream data structures in Redis and illustrate how to set up Redis streams
- retrieve data from a Redis stream using the XREAD command
- coordinate messaging between a collection of Redis clients by setting up a consumer group
- contrast messages that have been sent with those that are pending in the context of Redis consumer groups

Objectives

- use the redis-py package to connect to a Redis server from a Python app
- use Pub/Sub in a Python app to subscribe to a messaging channel and retrieve messages
- adopt pipelines in a Python app to send batches of Redis commands to a Redis server
- illustrate the steps involved in defining and executing transactions in Redis
- apply the watch feature to monitor keys for changes in Redis while executing a transaction
- optimize mass insert operations in Redis by using the pipe mode feature of the Redis CLI
- illustrate the processes of client tracking and server-assisted client-side caching in Redis
- apply the broadcast mode for client tracking in order to reduce memory use at the Redis server
- synchronize access to a shared Redis resource by using Redis keys to implement a lock
- use the built-in lock available in the python-redis-lock package to ensure thread-safe access to a shared Redis resource

The Redis In-memory Data Store: Working With Redis Clients

Objectives

- create a new user with a specific set of permissions using ACL commands
- illustrate how permissions can be applied to users using ACL categories
- set up a duplicate copy of a Redis server as a precursor to creating a replica
- establish a master-slave relationship between a Redis server and its replica
- apply the least-recently-used cache replacement policy for a Redis server
- apply the least-frequently-used cache replacement policy for a Redis server

Final Exam: Software Architect

- adopt pipelines in a Python app to send batches of Redis commands to a Redis server
- compare list and set data structures and apply the Redis CLI commands that apply to sets
- coordinate messaging between a collection of Redis clients by setting up a consumer group
- create and update numeric data in Redis
- create a new user using ACL commands
- create a new user with a specific set of permissions using ACL commands
- describe the properties of sorted sets and illustrate how they contrast with regular Redis sets
- describe the steps involved in defining and executing transactions in Redis
- describe the types of data that can be stored in Redis and the properties of the different data structures
- describe the use cases of the Redis data store
- detail the features of list structures in Redis and illustrate the types of operations that can be performed with them
- detail the processes of client tracking and server-assisted client-side caching in Redis
- detail the use cases of the Redis data store
- distinguish between list and set data structures and apply the Redis CLI commands that apply to sets
- download and install Redis on your machine
- establish a master-slave relationship between a Redis server and its replica
- explain how permissions can be applied to users using ACL categories
- extract elements from Redis list individually
- extract elements from Redis list individually or in bulk
- identify the content and format of stream data structures in Redis and illustrate how to set up Redis streams
- identify the features of list structures in Redis and illustrate the types of operations that can be performed with them
- identify the processes of client tracking and server-assisted client-side caching in Redis
- identify the properties of sorted sets and illustrate how they contrast with regular Redis sets
- identify the types of data that can be stored in Redis and the properties of the different data structures
- identify the use cases of the Redis data store
- illustrate how permissions can be applied to users using ACL categories
- illustrate the processes of client tracking and server-assisted client-side caching in Redis
- illustrate the steps involved in defining and executing transactions in Redis
- implement a subscription to multiple Pub/Sub channels that match a pattern
- implement Pub/Sub in a Python app to subscribe to a messaging channel and retrieve messages
- implement set operations such as union, intersection, and set difference on Redis sets
- implement the built-in lock available in the python-redis-lock package to ensure thread-safe access to a shared Redis resource
- implement the redis-py package to connect to a Redis server from a Python app
- install and use Redis on your machine
- install Redis on your machine
- invoke create, read, update and delete operations from the Redis CLI
- modify the elements in a sorted set
- optimize mass insert operations in Redis by using the pipe mode feature of the Redis CLI
- perform set operations such as union, intersection, and set difference on Redis sets
- publish to Pub/Sub channels using the Redis CLI
- read data from a Redis stream using the XREAD command
- recognize how Redis supports connections via various clients
- recognize the content and format of stream data structures in Redis and illustrate how to set up Redis streams
- recognize the features of list structures in Redis and illustrate the types of operations that can be performed with them
- remove and modify the elements in a sorted set
- retrieve data from a Redis stream using the XREAD command
- set up a duplicate copy of a Redis server as a precursor to creating a replica
- set up a master-slave relationship between a Redis server and its replica
- set up a subscription to multiple Pub/Sub channels that match a pattern
- subscribe to Pub/Sub channels using the Redis CLI
- update numeric data in Redis
- use ACL commands to create a new user with a specific set of permissions
- use a duplicate copy of a Redis server as a precursor to creating a replica
- use mass insert operations in Redis by using the pipe mode feature of the Redis CLI
- use pipelines in a Python app to send batches of Redis commands to a Redis server
- use Pub/Sub in a Python app to subscribe to a messaging channel and retrieve messages
- use the built-in lock available in the python-redis-lock package to ensure thread-safe access to a shared Redis resource
- use the Redis hash structure to extract data in the form of a map
- use the Redis hash structure to store and extract data in the form of a map
- use the redis-py package to connect to a Redis server from a Python app

Productivity Tools for Software Architects

Optional

COURSE

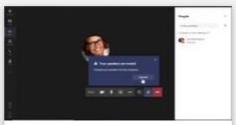
Working with Apps, Tabs & Wiki

418

COURSE

Making calls, Organizing Contacts & Using Voicemail

413



COURSE

Creating, Joining & Managing Meetings

408

COURSE

Signing in & Setting Up Slack

12

COURSE

Using Channels in Slack

4

COURSE

Using Private Messaging & Communication Tools in...

5

COURSE

Creating, Finding & Sharing Information in Slack

4

COURSE

Configuring Slack

2

COURSE

Creating & Setting Up Projects

92

COURSE

Configuring & Managing Boards

62

COURSE

Planning & Working on a Software Project

47

COURSE

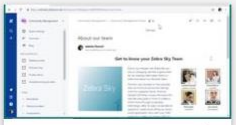
Reporting in Jira Software

49

COURSE

Signing in & Navigating within Spaces

24



COURSE

Setting Up & Managing Spaces

22

COURSE

Working with Spaces

14

COURSE

Working with Team Members

50

COURSE

Configuring Spaces

14

Bookshelf

Optional

BOOK

Java: The Complete Reference, Eleventh Edition

23

BOOK

JAVA Programming Simplified: From Novice to...

6

BOOK

Introducing Play Framework: Java Web Application...

BOOK

Java Program Design: Principles, Polymorphism,...

18

BOOK

Data Structures and Program Design Using Java: A Self-...

BOOK

Data Structures and Algorithms in Java, Sixth...

16

BOOK

Advanced Topics in Java: Core Concepts in Data...

6

BOOK

Data Structures Using Java

1

BOOK

Java Threads and the Concurrency Utilities

4

BOOK

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

3

BOOK

Pro JPA 2 in Java EE 8: An In-Depth Guide to Java...

BOOK

Spring Boot Persistence Best Practices: Optimize Java...

3

BOOK

Rapid Java Persistence and Microservices: Persistence...

2

BOOK

Java Persistence with Hibernate, Second Edition

BOOK

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

1

BOOK

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

2

BOOK

Practical Microservices Architectural Patterns:...

12

BOOK

Spring Boot Intermediate Microservices: Resilient...

19

BOOK

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

7

BOOK

Beginning Spring Boot 2: Applications and...

36

BOOK

Spring Boot and Single-Page Applications: Integrate You...

2

BOOK

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

18

BOOK

Pro Java Clustering and Scalability: Building Real-...

8

Business & Leadership for Software Architects

Optional

COURSE

Developing and Supporting an Agile Mind-set

593

COURSE

Encouraging Team Communication and...

827

COURSE

The Essential Role of the Agile Product Owner

182

COURSE

Using Strategic Thinking to Consider the Big Picture

393

COURSE

Getting to the Root of a Problem

639

COURSE

Unleashing Personal and Team Creativity

564

COURSE

Contributing as a Virtual Team Member

1667

COURSE

Developing a Growth Mind-set

1542

COURSE

Effective Team Communication

696

COURSE

Reaching Sound Conclusions

192

FOLLOW US ON:

www.skilltech.pl

email: biuro@skilltech.pl

tel. +48 22 44 88 827

SkillTech
Technology hired for excellence