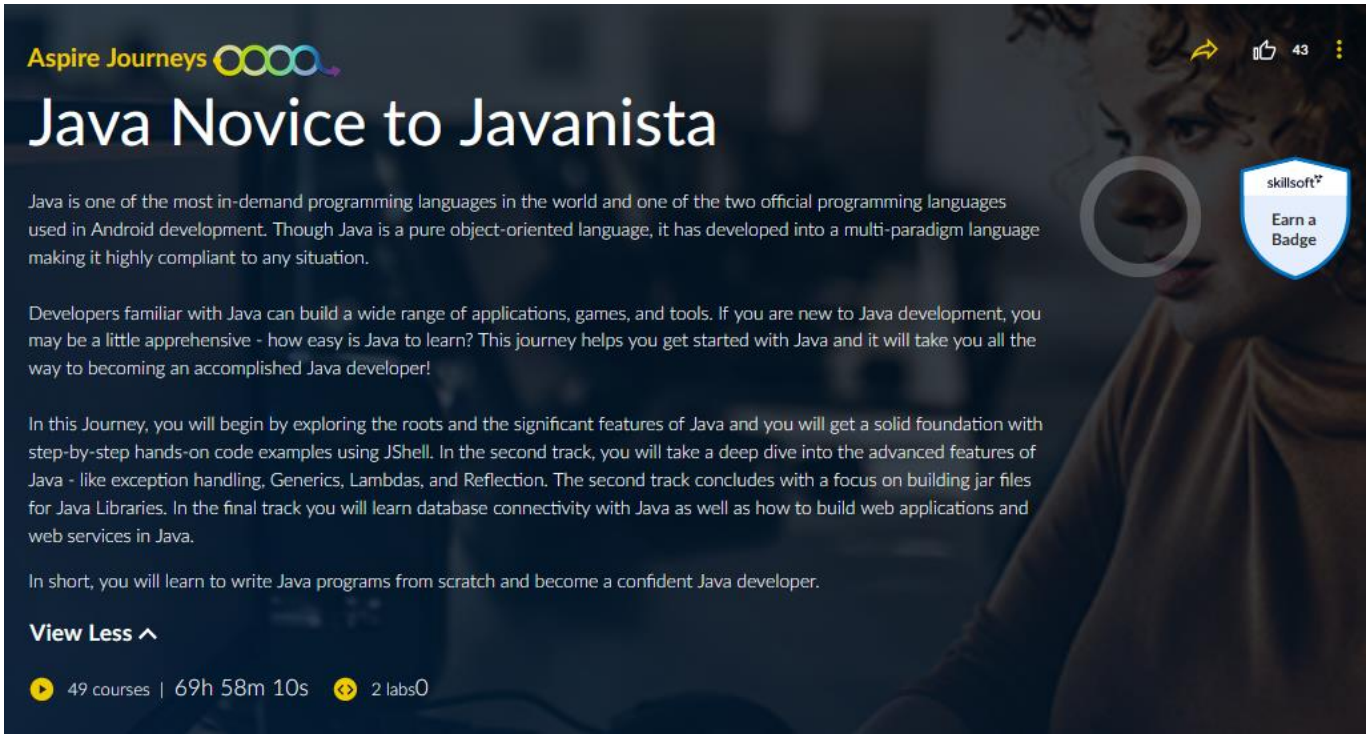
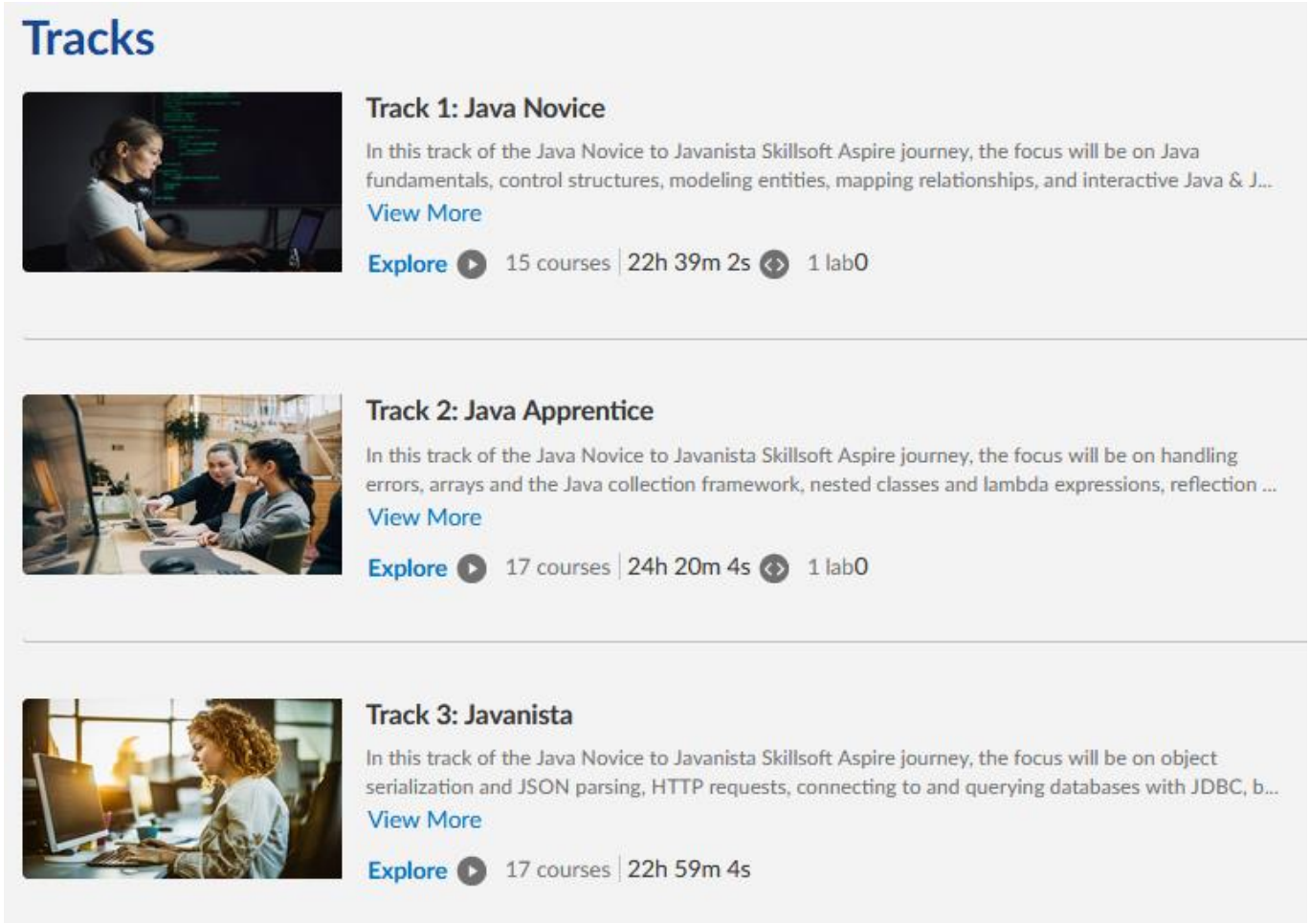
Java Novice to Javanista

SKILLSOFT ASPIRE JOURNEY

skillsoft

percipio™





PREREQUISITES

In order to fully profit from the potential of this Aspire Journey we recommend the following prerequisite skills:

- Programming knowledge
- Familiar with Object Oriented Programming
- Familiar with database concepts

Aspire Journeys: Java Novice to Javanista

Track 1: Java Novice

In this track of the Java Novice to Javanista Skillsoft Aspire journey, the focus will be on Java fundamentals, control structures, modeling entities, mapping relationships, and interactive Java & JShell.

15 courses | 22h 39m 2s 1 lab0

skillsoft
Earn a Badge

Getting Started with Java: The Fundamentals of Java Programming

Objectives

- recall what's meant when we say that Java is object-oriented and platform-independent
- describe how the JVM is used to execute your Java source code
- identify the purpose of the JVM and some of the important components in the Java ecosystem
- recognize how programming is affected by the static typing requirement in Java and the effects of its just-in-time compiler
- describe the workings of Java's automatic garbage collector
- enumerate the various use cases of Java in different fields of computer science

Getting Started with Java: Writing & Running Java Programs

Objectives

- install the Java Development Kit and the IntelliJ IDE on Microsoft Windows
- install and set up the Java Development Kit and the IntelliJ IDE on macOS
- create and open up a new Java project and a new class in IntelliJ
- set up a basic class with a main() method and run it from IntelliJ
- recognize some of the fundamental syntactic rules in Java
- add comments to your Java program to describe the code

Getting Started with Java: Variables & Primitive Types

Objectives

- recall how variables can be declared and created in Java
- publish variable values to the console using the format method in Java
- describe the use of various primitive Java data types, such as int and Boolean
- recognize the limits of different Java data types in terms of what values they can store
- identify the use of the byte, char, and short data types in Java
- describe how composite data can be created and accessed in the form of arrays in Java

Getting Started with Java: Operators

Objectives

- use Java math operators to perform operations on integers and concatenate strings
- recognize how Java assignment operators are used to change the values of variables
- describe the use of logical operators in Java to specify conditions and create filters
- make use of comparison operators and functions in Java to compare variables
- recall how arithmetic operations in Java follow the PEMDAS rules when evaluating expressions
- describe how various operations, such as trimming, are performed on Java strings
- use null values in Java to specify that a variable does not have any value
- implement Java casting operations to convert data from one type to another

Control Structures in Java: Implementing Java Control Structures

Objectives

- accept user input and implement conditional execution using if statements
- recognize the syntax of if statements and make use of else blocks
- use multiple if statements and explore the use of else-if blocks
- demonstrate how for loops can be used to iterate the contents of a string or array
- use switch statements to select categories based on a value
- describe the different ways in which for loops can be applied, such as nested for loops
- create a for loop and use the break and continue keywords to control its flow
- use while loops to perform operations while a condition is true
- manually compile and run Java code from the command line
- develop and run a Java program that accepts input from the command line

Modeling Entities in Java: Getting Started with Classes & Objects

Objectives

- recall how a Java class is a template with attributes and actions
- recognize that classes are blueprints and objects are instances of classes
- identify the difference between static and instance variables
- create a new Java class within an IntelliJ project
- set up an IntelliJ run configuration and instantiate class objects

Modeling Entities in Java: Defining Custom Classes & Objects

Objectives

- instantiate objects of built-in Java classes
- create objects in Java and access object information
- perform operations on Java objects
- create member variables in Java and view their values
- create constructors and use them to instantiate objects in Java
- define constructors and use them to initialize member variables in Java
- create, access, and view private member variables in Java
- define and invoke private methods in Java
- explore type safety in getter methods and edit member fields with setter methods in Java

Modeling Entities in Java: Methods, Method Overloading, & Constructors

Objectives

- implement return values and types correctly
- create objects of a Java class and use getters and setters to edit fields
- overload methods with the same name in Java
- recall how overloaded methods work and how they can invoke one another
- use the default, no-argument constructor to create objects
- initialize fields in constructor definitions
- create and use parameterized constructors in Java
- create multiple constructors with different signatures
- initialize fields using the 'this' keyword
- reuse code using constructor chaining
- summarize the key concepts covered in

Modeling Entities in Java: Static Members, Arguments, & Method Overriding

Objectives

- recall Java best practices to access static variables
- use class references to access static variables and describe the use of the 'final' keyword
- outline access restrictions on instance members from static members
- implement a scenario using a static variable to count instantiated objects
- use class reference to invoke and execute static methods
- reassign variables inside methods and explore where the changes are visible
- contrast the effects of variable reassignment with variable modification
- outline pass-by-value and pass-by-reference with custom objects
- recall the operation of the == operator and the default .equals() method
- examine how to work with the .equals() and .hashCode() methods and the hashCode contract
- override .equals() to check for semantic equality
- override .hashCode() and .equals() to honor the hashCode contract

Mapping Relationships in Java: Modeling Is-a Relationships Using Inheritance

Objectives

- outline how classes and inheritance can be used to model relationships
- model is-a relationships using the inheritance hierarchy
- recall how inheritance, polymorphism, and encapsulation are used in Java
- identify the existence of the universal Object base class
- examine methods inherited from the universal Object base class
- implement custom and built-in classes derived from the Object base class
- use the instanceof operator to identify types for objects
- identify the compile-time and runtime types for objects
- store objects of a derived class type in a variable of the base class type
- set up an inheritance hierarchy using custom classes
- identify the roles of the type of variable and type of object contained in the variable
- perform up-casting and down-casting operations on objects
- examine the inheritance of fields and methods in derived classes

Mapping Relationships in Java: Constructors & Polymorphism

Objectives

- examine how the default constructor in the base class is used in the derived class
- use the super keyword to invoke a base class constructor from a derived class
- invoke base class constructors correctly from derived classes
- investigate nuances while defining derived class constructors
- reuse code using the super() and this() constructor invocations
- recall the characteristics of runtime polymorphism
- invoke the right methods based on the runtime types of objects
- outline how runtime polymorphism uses dynamic method dispatch
- identify compile-time polymorphism with overloaded methods
- overload methods with the same name and different number and types of input arguments
- recall how Java identifies and invokes the right overloaded method based on parameter types
- perform type promotions and coercions in custom class inheritance hierarchies

Mapping Relationships in Java: Overriding Methods and Using Access Modifiers

Objectives

- override methods to have different method implementations in derived classes
- use the super keyword in overridden methods
- identify method hiding when redefining static methods with the same name
- use the final modifier to restrict a field from being initialized except in constructors
- use the final modifier with methods to prevent overriding and with classes to prevent inheritance
- list the characteristics of an abstract class
- use abstract base classes in inheritance hierarchies
- use the private access modifier to restrict access to fields and methods
- contrast the restrictions offered by the public and private access modifiers
- use factory methods to instantiate objects with private constructors
- use the protected access modifier to restrict access to derived classes and classes in the same package
- investigate the nuances of the protected access modifier
- use the default modifier to restrict access to other classes in the same package

Mapping Relationships in Java: Working with Interfaces & Class Loaders

Objectives

- recall the characteristics of interfaces and interface methods
- implement interface methods in class definitions
- define fields in an interface and recall the modifiers applied to these fields
- identify Java enforcements and checks for interface method implementations
- specify default implementations for interface methods
- define variables using interface types
- recall the use case for class loaders and list the class loader hierarchy
- access and use class loaders in Java
- use class loaders to manually load classes
- create a custom class loader with the right method implementations
- use a custom class loader to load user-defined classes

Interactive Java & JShell: Writing Java Programs with the Interactive JShell

Objectives

- set up JShell and execute basic Java commands in the interactive environment
- create and invoke functions in the JShell
- define a function that includes a forward reference and recognize the limitations of such methods
- adjust feedback modes to regulate the output produced after running code snippets
- use the auto-complete feature to automatically fill in code details
- recognize how imports work in JShell and how objects can be instantiated
- create and use classes or types in JShell in order to work with bespoke entities
- describe how objects are affected when the classes they are built out of are modified
- read in data from a CSV file into a JShell program
- export and save data in your program out to a file
- gather together snippets of previously executed code to create a JShell script

Final Exam: Java Novice

Objectives

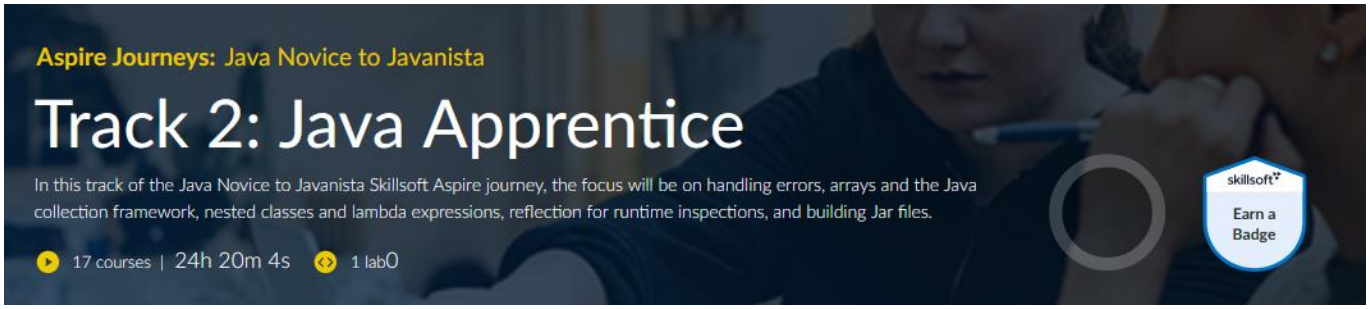
- accept user input and implement conditional execution using if statements
- access and use class loaders in Java
- contrast the effects of variable reassignment with variable modification
- contrast the restrictions offered by the public and private access modifiers
- create and invoke functions in the JShell
- create and use parameterized constructors in Java
- create constructors and use them to instantiate objects in Java
- create member variables in Java and view their values
- define a function that includes a forward reference and recognize the limitations of such methods
- define variables using interface types
- demonstrate how for loops can be used to iterate the contents of a string or array
- describe how the JVM is used to execute your Java source code
- describe the use of logical operators in Java to specify conditions and create filters
- describe the use of various primitive Java data types, such as int and Boolean
- enumerate the various use cases of Java in different fields of computer science
- examine how to work with the .equals() and .hashCode() methods and the hashCode contract
- explore nuances while defining derived class constructors
- explore the nuances of the protected access modifier
- identify the existence of the universal Object base class
- implement interface methods in class definitions
- implement return values and types correctly
- initialize fields in constructor definitions
- initialize fields using the 'this' keyword
- install and set up the Java Development Kit and the IntelliJ IDE on macOS
- install the Java Development Kit and the IntelliJ IDE on Microsoft Windows
- instantiate objects of built-in Java classes
- list the characteristics of an abstract class
- make use of comparison operators and functions in Java to compare variables
- model is-a relationships using the inheritance hierarchy
- overload methods with the same name and different number and types of input arguments
- override .equals() to check for semantic equality
- perform operations on Java objects
- publish variable values to the console using the format method in Java
- recall how a Java class is a template with attributes and actions
- recall how classes and inheritance can be used to model relationships
- recall how inheritance, polymorphism, and encapsulation are used in java
- recall how Java identifies and invokes the right overloaded method based on parameter types
- recall how the default constructor in the base class is used in the derived class
- recall how variables can be declared and created in Java
- recall Java best practices to access static variables
- recall the characteristics of interfaces and interface methods
- recall the characteristics of runtime polymorphism
- recall the operation of the == operator and the default .equals() method
- recall the use-case for class loaders and list the class loader hierarchy
- recognize how imports work in JShell and how objects can be instantiated
- recognize how Java assignment operators are used to change the values of variables
- recognize that classes are blueprints and objects are instances of classes
- recognize the syntax of if statements and make use of else blocks
- set up an inheritance hierarchy using custom classes
- set up JShell and execute basic Java commands in the interactive environment
- use class references to access static variables and describe the use of the 'final' keyword
- use Java math operators to perform operations on integers and concatenate strings
- use multiple if statements and explore the use of else-if blocks

- use switch statements to select categories based on a value
- use the auto-complete feature to automatically fill in code details
- use the default modifier to restrict access to other classes in the same package
- use the instanceof operator to identify types for objects
- use the private access modifier to restrict access to fields and methods
- use the protected access modifier to restrict access to derived classes and classes in the same package
- use the super keyword to invoke a base class constructor from a derived class

Java Novice

Objectives

- In this practice lab, learners will be presented with a series of exercises to practice developing in Java. Exercises include tasks working with operators, using control structures, working with classes and executing methods. Learners will also practice implementing inheritance, implementing constructors and polymorphism, and working with interfaces and Jshell.



Handling Errors: An Introduction to Exceptions

Objectives

- identify the different categories of exceptions in Java and how to deal with them
- outline the hierarchy of Exception classes in Java and recall specific exception types
- recognize compile errors in Java and distinguish these from exceptions
- define unchecked exceptions in the context of Java and how these can be acknowledged or handled by a developer
- list examples of checked exceptions in Java and contrast these with unchecked exceptions

Handling Errors: Handling Exceptions in Java

Objectives

- recognize how the throwing of exceptions influences the flow of a Java program
- implement a basic try-catch block to handle an exception in Java
- use multiple catch blocks to handle different types of exceptions in your code
- use a finally block to clean up after code execution in a try-catch
- use the throws keyword in a method signature instead of implementing an exception handler
- implement a try-with-resource block to close resources after their use

Handling Errors: Advanced Topics in Exceptions

Objectives

- use the throw keyword in Java to explicitly throw an exception when the state of the program does not match your own set of valid conditions
- invoke multiple exceptions in your program based on different error conditions
- throw and handle exceptions at various levels when nested function calls are involved
- define your exception class by extending Java's built-in Exception

Collections in Java: Arrays & Non-parameterized ArrayLists

Objectives

- create and add data to arrays of different types
- examine the fixed-length enforcement of arrays
- iterate over the values in arrays using for loops
- define functions to add and delete values in arrays
- use multidimensional arrays to store tabular data
- store and access elements in array lists, part of the Java collections framework
- work with type rules on non-parameterized collections

Collections in Java: Lists & List Operations

Objectives

- create and add data to arrays of different types
- examine the fixed-length enforcement of arrays
- iterate over the values in arrays using for loops
- define functions to add and delete values in arrays
- use multidimensional arrays to store tabular data
- store and access elements in array lists, part of the Java collections framework
- work with type rules on non-parameterized collections

Collections in Java: Sets & Maps

Objectives

- explore the basic characteristics of the 'set' data structure
- recall how hash sets identify duplicates
- recall how different types of sets are ordered
- implement tree sets for predictable ordering of data
- identify how the Comparator and Comparable interfaces are used in tree sets
- store key-value pairs in maps
- view keys and values in maps as collections
- create dictionaries with keys and values of custom types
- recognize different type of map implementations
- implement an LRU cache using a LinkedHashMap
- implement the Comparator interface to access elements in a tree map in priority order
- implement the Comparable interface to access elements in a tree map in priority order
- recall the special operations that can be performed on sorted maps

Generics in Java: Creating Classes and Methods Using Generics

Objectives

- list the advantages of writing generic code over non-generic code
- recall the limitations on code reuse in non-generic classes
- recall the limitations of using raw objects rather than generic types
- implement a class with generics for type safety and compile-time checks
- use generic types as input arguments to methods and return values from methods
- create non-parameterized objects from generic classes
- recall the disadvantages of non-generic methods
- parameterize the class definition to create generic methods
- parameterize the method definition without parameterizing classes
- illustrate type inference with parameterized methods

Generics in Java: Bounded Type Parameters & Wildcards

Objectives

- recall the disadvantages of unbounded type parameters
- constrain types using bounded type parameters
- specify type parameters with interface bounds
- use bounded type parameters with custom objects
- use multiple bounds with type parameters
- use upper-bounded wildcards
- compare and contrast upper-bounded wildcards and bounded type parameters
- recall when and how you would use unbounded wildcards
- use unbounded wildcards
- use lower-bounded wildcards
- recall how Java infers data types using wildcard capture
- recall the structure of valid type parameter names
- recall how the Java compiler uses type erasure with generic types

Classes in Java: Working with Static Nested, Inner, & Local Classes

Objectives

- illustrate how nested static classes work
- instantiate and use static nested classes
- illustrate various aspects of nested static classes
- recognize the limitations of static nested classes
- define and use inner classes
- access variables from inner classes
- use inner classes to create iterators for data structures
- define and use local classes
- access fields within local classes
- work with access modifiers and scopes in local classes
- define local classes in initialization blocks

Classes in Java: Creating & Using Anonymous Classes

Objectives

- define and use anonymous classes
- recognize how anonymous instances are objects
- illustrate the finer points of anonymous classes
- implement the built-in interfaces, Comparator and Runnable, with anonymous classes
- perform filter operations with local and anonymous classes

Classes in Java: Implementing Functional Interfaces Using Lambdas

Objectives

- define and use lambda expressions
- compare and contrast lambda statements and expressions
- recall how lambdas can only implement functional interfaces
- implement interfaces with generic types using lambdas
- mark functional interfaces with @FunctionalInterface annotation
- recall how functional interface instances are objects
- create Predicate, Consumer, Function, and Supplier interfaces
- invoke static and instance methods using method references
- invoke instance methods with types and invoke constructors using method references

Java: Getting Started with Reflection

Objectives

- create and set up a basic IntelliJ project to write java code
- define new classes and instantiate objects of these classes
- use class handles accessed via reflection to explore properties of classes
- view fields, methods, and their modifiers using reflection
- examine how you can identify classes from their objects
- use reflection to identify a variety of classes from objects
- get fully qualified names and simple names of classes using their handles
- investigate how you can get a handle to class objects using just the class name
- use reflection to identify the modifiers of class members
- identify anonymous, local, member classes, and interfaces using reflection
- identify enums, arrays, and primitives using reflection
- access package information, superclass, and declaring class information

Java: Accessing Constructors, Methods, & Fields Using Reflection

Objectives

- access the constructors in a class using reflection
- create objects of a class with handles to constructors
- access member variables of a class and its metadata
- access and update values of fields
- access and update protected and private fields
- access public, private, and protected methods
- view method parameters, annotations, return types, and exceptions
- view annotations on methods and recall the use of the retention policy
- observe how not all annotations are available at runtime for reflective access
- use method heuristics to identify getters and setters in a class
- invoke and call methods using handles

Java: Working with Annotations, Generics, & Arrays Using Reflection

Objectives

- examine what built-in annotations are accessible using reflection
- create a custom annotation and examine retention policies
- implement a use-case for reflection - accessing annotations to perform checks
- implement a use-case for reflection - using reflection to check the objects of classes for validity
- identify what generic information about type parameters is available using reflection
- explore the disadvantages of using reflection with generics
- view and edit data stored in arrays
- use reflective access to determine array types and component types

Java: Leveraging Reflection to Build Dynamic Proxies & Unit Tests

Objectives

- implement a dynamic proxy to create a dynamic object implementing an interface
- invoke interface methods and object base class methods on proxies
- pass proxy method invocations to a real object
- create annotations for the setup, teardown, and test case for a unit testing framework
- annotate test case methods for unit testing
- execute test cases using a custom test harness

Java Archive (JAR): Building Java Archives

Objectives

- use an integrated development environment (IDE) to create the main class for a Java project
- create and execute a basic Java archive using a manifest file
- extract and recognize the contents of a Java archive (JAR)
- outline the automatic generation of manifests and the fields contained in them
- edit and specify the data stored inside the manifest file of a JAR
- build a Java application with a dependency on an external library
- reference and use external dependencies in your JAR
- construct a JAR file with multiple main classes and run each of them
- create JAR files with multiple packages to store related code
- use non-executable JAR files as libraries in other projects

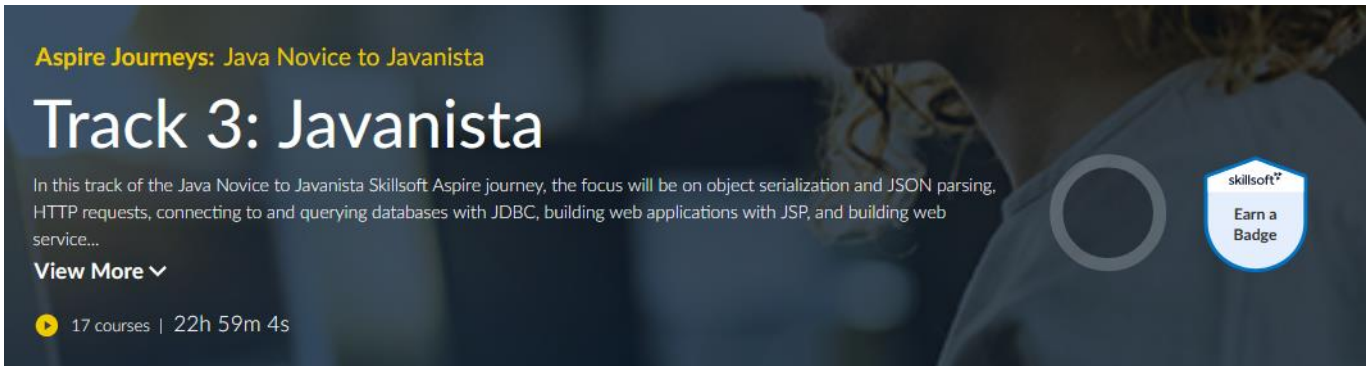
Java Archive (JAR): Packaging Java Apps Using Maven

Objectives

- install and configure Maven to create Java archives
- create a Maven project which can then be packaged into a JAR file
- create a custom pom.xml file and build an executable JAR with Maven
- execute and view the contents of a Java archive built with Maven
- create and run a Java archive that has external dependencies
- recognize how dependencies are referenced from a POM file
- create fat or uber JARs that package external dependencies in the archive using the Maven assembly plugin
- describe how dependencies are stored in fat or uber JARs
- build and execute a fat or uber JAR file created with the Maven Shade plugin

Java Apprentice

- In this practice lab, learners will be presented with a series of exercises to practice developing in Java. Exercises include tasks such as implemeting try-catch and finally blocks, performing operations on a list, implementing the Comparator Interface, and implementing classes with generic type parameters. Learners will also practice using bounded type parameters, working with static nested classes, using reflection and creating and executing a basic JAR file using Manifest file.



Serialization in Java: Getting Started with Object Serialization

Objectives

- set up the environment to perform hands-on coding
- serialize objects to byte streams and deserialize byte streams to objects
- serialize and deserialize custom, user-defined objects
- recall the role of the serialVersionUID field
- outline backward compatible and non-backward compatible changes
- use the transient modifier to omit fields from serialization
- serialize objects with nested object references
- send byte stream data over socket connections
- use the externalizable interface for control over serialization

Serialization in Java: Using JSON Simple for Serialization & Parsing

Objectives

- recall how the JSON format structures primitives, entities, and arrays
- set up an Apache Maven project and specify dependencies
- use JSON-simple to read in and write out JSON data
- use JSON-simple to read in and write out JSON arrays
- use JSON-simple to read in and write out complex JSON structures
- explore exceptions that arise when parsing JSON data
- parse JSON data using the SAX interface
- encode JSON data using strings and streams
- serialize custom classes to the JSON format

Serialization in Java: Using JSON in Java for Serialization & Parsing

Objectives

- use the org.json library for JSON parsing
- read and write JSON data using org.json
- import and export JSON arrays using org.json
- use the put(), accumulate(), and append() methods to work with arrays
- tokenize strings using the string tokener
- serialize custom classes to the JSON format with org.json
- serialize classes with nested objects with org.json
- use the @JSONPropertyName and @JSONPropertyIgnore annotations
- parse comma-delimited strings using the CDL class
- parse comma-delimited files and records using CDL

HTTP Requests in Java: Sending Simple HTTP Requests

Objectives

- set up a Maven project to build the HTTP client application
- use the HttpURLConnection to configure and send an HTTP GET request
- recognize various fields available in the response returned for an HTTP GET request
- transform the body of an HTTP response containing JSON data into a formatted JSON string
- include query parameters when submitting an HTTP GET request by adding them to the URL
- set up and send an HTTP POST request along with JSON data in the request body
- send HTTP PUT and DELETE requests to modify or delete a resource at the server
- describe the role of a HEAD request to obtain information about a resource
- set an upper limit on the amount of time you can wait for an HTTP request

HTTP Requests in Java: HTTP Requests with Java's HttpClient

Objectives

- use the HttpClient class to send a GET request and process the response that is returned
- describe the different methods available to access the data and metadata in an HttpResponse instance
- set an upper limit on the amount of time you can wait for a response to an HTTP request
- recognize the steps involved in automatic redirects and how these can be detected and handled with HttpClient
- send an asynchronous request using HttpClient and recognize how this differs from a synchronous request
- develop a program to send multiple asynchronous HTTP GET requests and process their responses once you are ready
- implement POST, PUT and DELETE request calls using HttpClient

Java Database Connectivity (JDBC): An Introduction to JDBC

Objectives

- set up your machine with a MySQL server and a client application to interact with it
- create a Maven project which uses the MySQL connector
- configure JDBC objects in order to connect to a database
- use a DataSource instance rather than a DriverManager in order to connect to a database from a Java app
- execute SELECT queries against a database using a JDBC Statement
- use a ResultSet in order to access and parse the results of a SELECT query
- create, load, and run parameterized SQL queries using the JDBC PreparedStatement
- set multiple parameters in a PreparedStatement in order to run INSERT queries
- execute UPDATE and DELETE queries and explore the data returned for each execution

Java Database Connectivity (JDBC): Interacting with Databases using RowSets

Objectives

- use a JdbcRowSet to connect to and run queries against a database and parse the results
- recognize some of the important settings of a JdbcRowSet and its default values
- move to different positions in a JdbcRowSet by using various navigation methods
- describe how the connected nature of a JdbcRowSet can be used to get any updates from underlying data
- update the content of existing rows in a database table using a JdbcRowSet
- add and delete rows from a database table using a JdbcRowSet
- describe the similarities between a JdbcRowSet and a CachedRowSet
- recognize the differences between a JdbcRowSet and a CachedRowSet
- update data in existing rows of a database table with a CachedRowSet
- perform insert and delete operations with a CachedRowSet

Java Database Connectivity (JDBC): Joining & Filtering Data with RowSets

Objectives

- create a number of related database tables, which can be joined based on common fields
- implement a join operation in a Java program using a JDBC JoinRowSet
- recognize the types of join operations that are supported by a JoinRowSet implementation
- join multiple tables involving different join fields using a JoinRowSet
- define the filtering conditions to be applied to a FilteredRowSet when implementing the Predicate class
- apply a predicate instance with a FilteredRowSet to apply a filter to a RowSet
- configure a predicate implementation to use multiple conditions when defining a filter

Java Database Connectivity (JDBC): Batch Executions & Transactions with JDBC

Objectives

- use a Statement object to run a series of INSERT queries as a batch
- perform a batch execution of parameterized queries using a PreparedStatement
- describe the behavior of a batch execution, where some queries are invalid
- recognize the limitations of running related SQL queries independently
- identify the steps involved in running a set of related queries as a transaction
- divide a large chunk of queries into units that can be committed to a database using Savepoints
- use Savepoints in batch executions and recognize the behavior of a program when this is done

Building Web Applications with JSP: An Introduction to JSP

Objectives

- download and set up the Apache Maven build automation tool
- use Apache Maven to build a stub web application
- set up Apache Tomcat web server that can be used to serve a JSP application
- create a WAR file for a web app and deploy it to a Tomcat web server
- develop a basic JSP app containing both HTML and Java code
- use a variety of JSP tags to define Java methods and invoke them in a JSP source file
- view various properties of an incoming request object
- unpack and view the contents of the WAR archive for a JSP application and describe its structure and properties
- build a composite JSP page by referencing external JSPs

Building Web Applications with JSP: Handling Errors

Objectives

- set up a JSP page to direct an end-user to a specific error page in the event of an exception being thrown
- add references to external resources in a JSP page by using the JSP expression language to get the application context path
- use a web.xml file to define a common error page for the entire JSP application
- adopt the Java try-catch block for exception handling and explore some of the recommended practices

Building Web Applications with JSP: Customizing Responses with Servlets

Objectives

- define a servlet class to process a simple GET request
- recognize the power of the WebServlet annotation when it comes to mapping servlets to URL endpoints
- use an HttpServletRequest object to access the parameters submitted by an end-user of a web application
- forward a request to a different endpoint in an application using the RequestDispatcher
- transfer processed data from a servlet to a JSP file for display in a web app
- recognize the power and scope of the ServletConfig and ServletContext classes when working with servlets
- set attributes at the request, session, and application-level using the corresponding setter methods
- describe the process of retrieving attributes from various scopes in your app, and recognize the limitations of each scope

Building Web Applications with JSP: Integrating a JSP App with a Database

Objectives

- set up MySQL server on your machine
- create a database schema and table and load it with data
- use JDBC to connect to and query a SQL database from a JSP app
- deploy and test the connectivity from your JSP app to a database and ensure that queried data is displayed correctly
- define a class representing an entity whose details are stored in the database and which an end-user of your app will work with
- use JDBC objects to query for single or multiple rows in a database table
- implement tags from the JSTL library to iterate over multiple rows of data
- execute an insert query from a Java application to add a new row of data to a SQL table
- automate the creation and population of a Java object with the useBean and setProperty tags in a JSP
- enable end-users to update existing rows in a database table via your JSP app
- allow end-users to delete rows from a database table through the UI of your JSP app

Java Web Services: Getting Started with SOAP-based Web Services

Objectives

- install Maven and use it to create a web service app
- explore the pom.xml file in a Maven project and set it up to use JAX-WS
- define a web service method that responds to a request by returning text
- describe the WSDL file for a web service and recognize its properties
- develop and run a client application that consumes a web service
- set up an endpoint interface for your web service application

Java Web Services: Integrating Web Services with a Database

Objectives

- create and configure a web service project that works with Java objects
- define a web service method that returns Java objects in a response
- develop a client app that requests for and processes Java objects in a SOAP response
- create a table in a database and connect to it from a Java app
- code a client application that requests a web service for data from a database
- define a web method that accepts a parameter from a client
- set up an input parameter of a custom type for a web method that adds data to a database
- set up a web method that performs a delete operation in a database
- create a web method that carries out an update operation in a database
- install and configure Apache Tomcat server
- deploy your SOAP-based application as a WAR file to a Tomcat server
- verify the behavior of a web service app deployed to a Tomcat server

Java Web Services: Building REST APIs

Objectives

- set up a web application with Maven and configure a POM file for a RESTful app
- use the Jersey framework to define a method that responds to GET requests
- consume a REST API by sending a GET request and processing its response
- configure return types in REST API methods
- define REST API methods that return Java objects in the form of JSON data
- recognize how the Jersey framework transforms Java objects to the JSON format

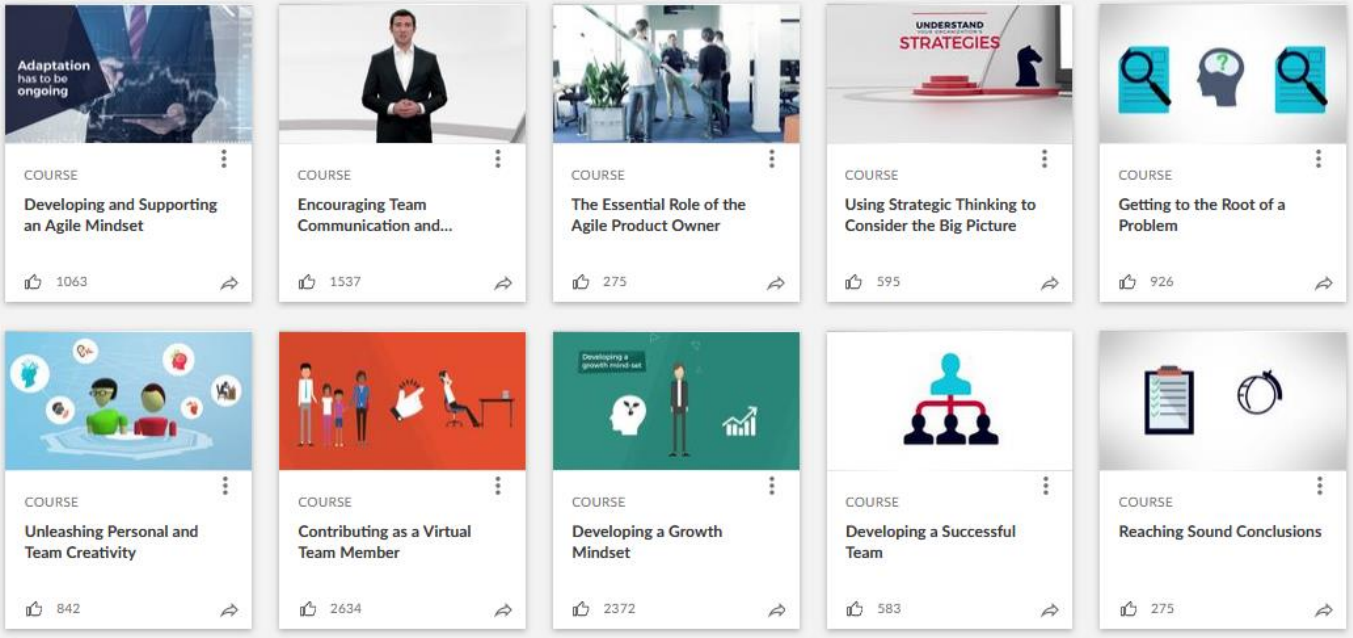
Java Web Services: Enabling CRUD Operations with REST APIs

Objectives

- connect to a database from a RESTful Java application
- send GET requests to a RESTful app for data stored in a database
- add records to a database table by means of a POST request
- update data on a database by implementing a PUT request
- delete data on a database by processing a DELETE request

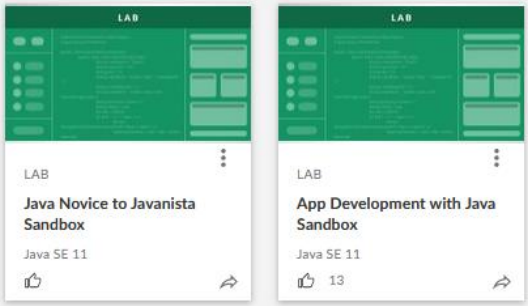
Business & Leadership for Javanista

Optional



Additional Resources

Optional



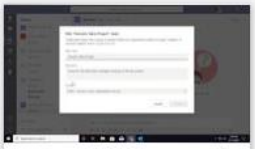
Productivity Tools for Javanista

Optional

COURSE

Getting to know the application

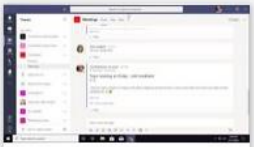
1009



COURSE

Using Teams & Channels

746



COURSE

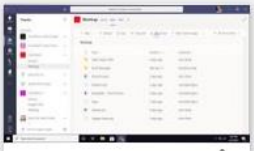
Communicating via the App

729

COURSE

Formatting, Illustrating & Reacting to Messages

542



COURSE

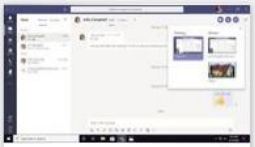
Creating, Finding & Organizing Files

536

COURSE

Working with Apps, Tabs & Wiki

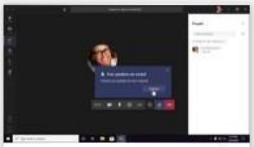
465



COURSE

Making calls, Organizing Contacts & Using Voicemail

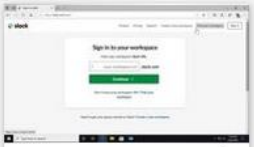
453



COURSE

Creating, Joining & Managing Meetings

462



COURSE

Signing in & Setting Up Slack

27

COURSE

Using Channels in Slack

12

COURSE

Using Private Messaging & Communication Tools in...

14

COURSE

Creating, Finding & Sharing Information in Slack

8

COURSE

Configuring Slack

7

COURSE

Creating & Setting Up Projects in Jira Cloud

164

COURSE

Configuring & Managing Boards in Jira Cloud

105

COURSE

Planning & Working on a Software Project in Jira...

85

COURSE

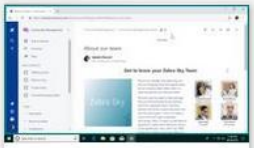
Reporting in Jira Software

83

COURSE

Signing in & Navigating within Spaces

40



COURSE

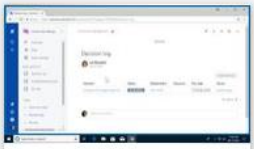
Setting Up & Managing Spaces

33

COURSE

Working with Space

27



COURSE

Working with Team Members

75

COURSE

Configuring Spaces

20

Bookshelf

Optional

BOOK

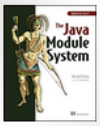
JAVA Programming Simplified: From Novice to...

21

BOOK

Java 13 Revealed: For Early Adoption and Migration,...

4



BOOK

The Java Module System

3

BOOK

Modern Java in Action: Lambdas, Streams, Reactive...

27

BOOK

Functional Programming in Java: How Functional...

8

BOOK

Java in Easy Steps, 7th Edition

9

BOOK

Java for Absolute Beginners: Learn to Program the...

112

BOOK

Data Structures and the Java Collections Framework, Thi...

20

BOOK

Java Language Features: With Modules, Streams,...

3

BOOK

Java 8 in Action: Lambdas, Streams, and Functional-...

13

BOOK

Java XML and JSON: Document Processing for...

6

BOOK

Servlet & JSP: A Beginner's Tutorial

5

BOOK

R2DBC Revealed: Reactive Relational Database...