**Understanding Java Technology and environment**

* Describe Java Technology and the Java development
* Identify key features of the Java language

**Working With Java Primitive Data Types and String APIs**

* Declare and initialize variables (including casting and promoting primitive data types)
* Identify the scope of variables
* Use local variable type inference
* Create and manipulate Strings
* Manipulate data using the StringBuilder class and its methods

**Working with Java Arrays**

* Declare, instantiate, initialize and use a one-dimensional array
* Declare, instantiate, initialize and use two-dimensional array

**Creating and Using Methods**

* Create methods and constructors with arguments and return values
* Create and invoke overloaded methods
* Apply the static keyword to methods and fields

**Reusing Implementations Through Inheritance**

* Create and use subclasses and superclasses
* Create and extend abstract classes
* Enable polymorphism by overriding methods
* Utilize polymorphism to cast and call methods, differentiating object type versus reference type
* Distinguish overloading, overriding, and hiding

**Handling Exceptions**

* Describe the advantages of Exception handling and differentiate among checked, unchecked exceptions, and Errors
* Create try-catch blocks and determine how exceptions alter program flow
* Create and invoke a method that throws an exception

**Creating a Simple Java Program**

* Create an executable Java program with a main class
* Compile and run a Java program from the command line
* Create and import packages

**Using Operators and Decision Constructs**

* Use Java operators including the use of parentheses to override operator precedence
* Use Java control statements including if, if/else, switch
* Create and use do/while, while, for and for each loops, including nested loops, use break and continue statements

**Describing and Using Objects and Classes**

* Declare and instantiate Java objects, and explain objects' lifecycles (including creation, dereferencing by reassignment, and garbage collection)
* Define the structure of a Java class
* Read or write to object fields

**Applying Encapsulation**

* Apply access modifiers
* Apply encapsulation principles to a class

**Programming Abstractly Through Interfaces**

* Create and implement interfaces
* Distinguish class inheritance from interface inheritance including abstract classes
* Declare and use List and ArrayList instances
* Understanding Lambda Expressions

**Understanding Modules**

* Describe the Modular JDK
* Declare modules and enable access between modules
* Describe how a modular project is compiled and run

**Java Fundamentals**

* Create and use final classes
* Create and use inner, nested and anonymous classes
* Create and use enumerations

**Java Interfaces**

* Create and use interfaces with default methods
* Create and use interfaces with private methods

**Functional Interface and Lambda Expressions**

* Define and write functional interfaces
* Create and use lambda expressions including statement lambdas, local-variable for lambda parameters

**Built-in Functional Interfaces**

* Use interfaces from java.util.function package
* Use core functional interfaces including Predicate, Consumer, Function and Supplier
* Use primitive and binary variations of base interfaces of java.util.function package

**Migration to a Modular Application**

* Migrate the application developed using a Java version prior to SE 9 to SE 11 including top-down and bottom-up migration, splitting a Java SE 8 application into modules for migration
* Use jdeps to determine dependencies and identify way to address the cyclic dependencies

**Concurrency**

* Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks
* Use java.util.concurrent collections and classes including CyclicBarrier and CopyOnWriteArrayList
* Write thread-safe code
* Identify threading problems such as deadlocks and livelocks

**I/O (Fundamentals and NIO2)**

* Read data from and write console and file data using I/O Streams
* Use I/O Streams to read and write files
* Read and write objects by using serialization
* Use Path interface to operate on file and directory paths
* Use Files class to check, delete, copy or move a file or directory
* Use Stream API with Files

**Database Applications with JDBC**

* Connect to databases using JDBC URLs and DriverManager
* Use PreparedStatement to perform CRUD operations
* Use PreparedStatement and CallableStatement APIs to perform database operations

**Annotations**

* Describe the purpose of annotations and typical usage patterns
* Apply annotations to classes and methods
* Describe commonly used annotations in the JDK
* Declare custom annotations

**Exception Handling and Assertions**

* Use try-with-resources construct
* Create and use custom exception classes
* Test invariants by using assertions

**Generics and Collections**

* Use wrapper classes, auto-boxing and auto-unboxing
* Create and use generic classes, methods with diamond notation and wildcards
* Describe Collections Framework and use key collection interfaces
* Use Comparator and Comparable interfaces
* Create and use convenience methods for collections

**Java Stream API**

* Describe the Stream interface and pipelines
* Use lambda expressions and method references

**Lambda Operations on Streams**

* Extract stream data using map, peek and flatMap methods
* Search stream data using search findFirst, findAny, anyMatch, allMatch and noneMatch methods
* Use Optional class
* Perform calculations using count, max, min, average and sum stream operations
* Sort a collection using lambda expressions
* Use Collectors with streams, including the groupingBy and partitioningBy operation

**Services in a Modular Application**

* Describe the components of Services including directives
* Design a service type, load services using ServiceLoader, check for dependencies of the services including consumer and provider modules

**Parallel Streams**

* Develop the code that use parallel streams
* Implement decomposition and reduction with streams

**Secure Coding in Java SE Application**

* Prevent Denial of Service in Java applications
* Secure confidential information in Java application
* Implement Data integrity guidelines- injections and inclusion and input validation
* Prevent external attack of the code by limiting Accessibility and Extensibility, properly handling input validation, and mutablity
* Secure constructing sensitive objects
* Secure Serialization and Deserialization

**Localization**

* Use Locale class
* Use resource bundles
* Format messages, dates, and numbers with Java