Author: Srinivas D

Contents:

Author: Srinivas D

1. Module 1: Introduction to Java

- Overview of Java programming language
- Setting up Java development environment
- Writing and executing basic Java programs
- Introduction to variables, data types, and operators in Java

Key Topics:

Overview of Java Programming:

- Introduction to Java as a high-level, object-oriented programming language.
- Brief history of Java and its significance in the software industry.
- Advantages and use cases of Java compared to other programming languages.

Setting up Java Development Environment:

- Installing Java Development Kit (JDK) on different operating systems (Windows, macOS, Linux).
- Configuring environment variables such as JAVA_HOME and PATH.
- Installing and setting up Integrated Development Environments (IDEs) like IntelliJ IDEA, Eclipse, or NetBeans.

Writing and Running Basic Java Programs:

- Understanding the structure of a Java program: classes, methods, and statements.
- Writing simple "Hello, World!" program in Java to understand the syntax.
- Compiling and executing Java programs using command-line tools or IDEs.

Variables, Data Types, and Operators:

- Introduction to variables and data types in Java (int, double, boolean, etc.).
- Understanding variable declaration, initialization, and assignment.
- Overview of arithmetic, relational, logical, and assignment operators in Java.

Learning Objectives:

- Gain a foundational understanding of the Java programming language.
- Learn how to set up a Java development environment on different platforms.
- Acquire practical skills in writing, compiling, and running basic Java programs.

Author: Srinivas D

- Understand fundamental concepts such as variables, data types, and operators in Java.

Instructional Methods:

- Lectures: Detailed explanations of Java programming concepts, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Quizzes and Assessments: Regular quizzes and assessments to evaluate understanding and retention of key concepts.
- Project Work: Small projects or tasks to apply knowledge gained in real-world scenarios.

Assessment:

- Programming Assignments: Completion of coding assignments to demonstrate understanding of basic Java syntax and concepts.
- Quizzes and Tests: Regular quizzes and tests to assess understanding of key topics covered in the module.
- Project Work: Evaluation of project work to assess practical application of Java programming skills.

2. Module 2: Object-Oriented Programming in Java

- Understanding classes and objects
- Encapsulation, inheritance, and polymorphism
- Constructors and method overloading/overriding.
- Abstract classes and interfaces

Key Topics:

Classes and Objects:

- Understanding the concept of classes and objects in Java.
- Creating and using objects to model real-world entities.

- Defining classes with attributes (fields) and behaviors (methods).

Encapsulation:

- Understanding encapsulation as a fundamental principle of OOP.
- Using access modifiers (public, private, protected) to control access to class members.
- Encapsulating data and behavior within Java classes to ensure data integrity.

Inheritance:

- Understanding inheritance and its benefits in code reuse and extensibility.
- Creating class hierarchies and subclassing in Java.
- Overriding methods and accessing superclass members.

Polymorphism:

- Understanding polymorphism and its three forms: compile-time, runtime, and method overriding.
- Implementing polymorphism in Java using method overriding and dynamic method dispatch.
- Leveraging polymorphism to write flexible and maintainable code.

Abstract Classes and Interfaces:

- Defining abstract classes and methods in Java.
- Implementing interfaces to achieve abstraction and multiple inheritance.
- Understanding the difference between abstract classes and interfaces and when to use each.

Learning Objectives:

- Understand the principles of Object-Oriented Programming (OOP) and how they apply to Java.
- Gain proficiency in creating and using classes and objects in Java programs.
- Learn about encapsulation, inheritance, polymorphism, abstract classes, and interfaces in Java.

Instructional Methods:

- Lectures: Detailed explanations of OOP concepts in Java, accompanied by examples and illustrations.

- Hands-on Practice: Interactive coding exercises and assignments to reinforce understanding and application of OOP principles.
- Code Reviews: Peer or instructor-led code reviews to evaluate and provide feedback on participants' implementation of OOP concepts.
- Case Studies: Analysis of real-world scenarios where OOP principles are applied in Java programming.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in applying OOP principles in Java programs.
- Quiz and Tests: Regular quizzes and tests to assess understanding of OOP concepts covered in the module.
- Project Work: Evaluation of project work to assess practical application of OOP principles in Java programming.

3. **Module 3: Exception Handling and Debugging**

- Handling exceptions in Java: try-catch blocks.
- Checked vs. unchecked exceptions.
- Using try-with-resources statement.
- Debugging techniques in Java: breakpoints, watches, and stack traces

Key Topics:

Understanding Exceptions:

- Introduction to exceptions and error handling in Java.
- Different types of exceptions: checked exceptions, unchecked exceptions, and errors.
- The try-catch-finally construct for handling exceptions.

Exception Handling Techniques:

- Using try-catch blocks to handle exceptions gracefully.
- Throwing and propagating exceptions using the throw and throws keywords.
- Multi-catch blocks and try-with-resources statements (Java 7+).

Checked vs. Unchecked Exceptions:

- Understanding the distinction between checked and unchecked exceptions.
- Handling checked exceptions with try-catch blocks or declaring them in method signatures.
- Dealing with unchecked exceptions and errors in Java programs.

Debugging Java Programs:

- Introduction to debugging and its importance in software development.
- Using breakpoints to pause execution and inspect variables.
- Stepping through code: stepping into, stepping over, and stepping out of methods.

Using Debugging Tools:

- Overview of debugging tools available in popular Java IDEs (IntelliJ IDEA, Eclipse, NetBeans).
- Analyzing stack traces and exception messages to identify the cause of errors.
- Using debugging features like watches, expression evaluation, and conditional breakpoints.

Learning Objectives:

- Understand the principles of exception handling in Java and how to use try-catch blocks effectively.
- Learn how to differentiate between checked and unchecked exceptions and handle them appropriately.
- Gain proficiency in debugging Java programs using breakpoints, stack traces, and debugging tools.

Instructional Methods:

- Lectures: Detailed explanations of exception handling concepts and debugging techniques, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and debugging tasks to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' exception handling and debugging skills.
- Case Studies: Analysis of real-world scenarios where effective exception handling and debugging techniques are applied.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in writing robust exception handling code and debugging Java programs.
- Quiz and Tests: Regular quizzes and tests to assess understanding of exception handling concepts and debugging techniques covered in the module.
- Project Work: Evaluation of project work to assess practical application of exception handling and debugging skills in Java programming.

4. Module 4: Collections Framework

- Overview of the Java Collections Framework (JCF)
- Lists, Sets, and Maps interfaces
- Common implementations: ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap
- Iterating through collections and performing operations

Key Topics:

Introduction to Collections Framework:

- Overview of the Java Collections Framework (JCF) architecture and hierarchy.
- Understanding the core interfaces: List, Set, Queue, and Map.
- Benefits and advantages of using collections in Java programming.

Lists and their Implementations:

- Understanding the List interface and its implementations: ArrayList, LinkedList, and Vector.
- Differences between ArrayList and LinkedList in terms of performance and usage scenarios.
- Common operations and methods for manipulating lists: add, remove, get, and iterate.

Sets and their Implementations:

- Understanding the Set interface and its implementations: HashSet, TreeSet, and LinkedHashSet.
- Characteristics of sets: uniqueness, ordering, and performance considerations.
- Performing set operations such as union, intersection, and difference.

Maps and their Implementations:

- Understanding the Map interface and its implementations: HashMap, TreeMap, and LinkedHashMap.
- Working with key-value pairs and associating values with keys.
- Common operations and methods for manipulating maps: put, get, remove, and iterate.

Iterating through Collections:

- Different ways to iterate through collections: using iterators, enhanced for loop, and forEach method.
- Understanding the iterator pattern and its usage in Java collections.
- Using lambda expressions and functional interfaces with forEach method (Java 8+).

Learning Objectives:

- Understand the Java Collections Framework and its core interfaces.
- Gain proficiency in using different types of collections (lists, sets, maps) and their implementations.
- Learn how to perform common operations and iterate through collections effectively.

Instructional Methods:

- Lectures: Detailed explanations of collection framework concepts, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' usage of collections and provide feedback.

- Case Studies: Analysis of real-world scenarios where collections are used to solve programming problems.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in using collections to solve programming tasks.
- Quiz and Tests: Regular quizzes and tests to assess understanding of collection framework concepts and operations.
- Project Work: Evaluation of project work to assess practical application of collections framework in Java programming.

5. **Module 5: Generics and Type Safety**

- Introduction to generics in Java
- Writing generic classes and methods
- Using wildcards in generics
- Ensuring type safety with generics

Key Topics:

Introduction to Generics:

- Understanding the need for generics in Java.
- Overview of generic types, methods, and classes.
- Benefits of using generics: type safety, code reusability, and improved readability.

Defining Generic Classes and Methods:

- Writing generic classes with type parameters.
- Creating generic methods with type parameters and type inference.
- Understanding type erasure and its implications for generics in Java.

Using Wildcards in Generics:

- Introduction to wildcard types (?).
- Understanding upper bounded wildcards (<? extends T>) and lower bounded wildcards (<? super T>).
- Using wildcard capture to work with unknown generic types.

Ensuring Type Safety with Generics:

- Leveraging generics to enforce compile-time type checking.
- Avoiding type casts and class cast exceptions with generics.
- Writing generic algorithms and data structures.

Learning Objectives:

- Understand the principles of generics in Java and their role in enhancing type safety and code reusability.
- Gain proficiency in defining and using generic classes, methods, and interfaces.
- Learn how to ensure type safety and avoid runtime errors using generics.

Instructional Methods:

- Lectures: Detailed explanations of generics concepts, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' usage of generics and provide feedback.
- Case Studies: Analysis of real-world scenarios where generics are used to solve programming problems.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in using generics to solve programming tasks.
- Quiz and Tests: Regular quizzes and tests to assess understanding of generics concepts and usage.
- Project Work: Evaluation of project work to assess practical application of generics in Java programming.

6. **Module 6: File Handling and I/O Operations**

- Reading and writing text files in Java

- Working with streams: InputStream, OutputStream, Reader, Writer
- Serialization and deserialization of objects
- File handling best practices and error handling

Key Topics:

Reading and Writing Text Files:

- Using FileReader and FileWriter to read from and write to text files.
- Understanding character encoding and decoding.
- Reading and writing text files line by line.

Working with Streams:

- Introduction to Java I/O streams: InputStream, OutputStream, Reader, and Writer.
- Reading and writing data using byte streams and character streams.
- Buffering streams for improved performance.

Serialization and Deserialization:

- Overview of object serialization and deserialization in Java.
- Implementing Serializable and Externalizable interfaces.
- Serializing and deserializing objects to/from files and streams.

File Handling Best Practices:

- Handling exceptions and error conditions in file I/O operations.
- Using try-with-resources statement for automatic resource management (Java 7+).
- Closing streams and releasing system resources properly.

Learning Objectives:

- Understand how to perform file handling and input/output operations in Java.
- Gain proficiency in reading from and writing to text files using FileReader and FileWriter.
- Learn how to work with streams, perform serialization and deserialization, and apply best practices in file handling.

Instructional Methods:

- Lectures: Detailed explanations of file handling concepts and I/O operations, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' file handling implementations and provide feedback.
- Case Studies: Analysis of real-world scenarios where file handling and I/O operations are used to solve programming problems.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in file handling and I/O operations.
- Quiz and Tests: Regular quizzes and tests to assess understanding of file handling concepts and best practices.
- Project Work: Evaluation of project work to assess practical application of file handling and I/O operations in Java programming.

7. **Module 7: Multithreading and Concurrency**

- Basics of multithreading in Java
- Creating and managing threads
- Synchronization techniques: synchronized keyword, locks, and conditions
- Java concurrency utilities: Executors, ThreadPoolExecutor, and Callable

Key Topics:

Introduction to Multithreading:

- Understanding threads and the multithreading model in Java.
- Benefits and challenges of multithreading in application development.
- Creating and managing threads using the Thread class and Runnable interface.

Synchronization and Thread Safety:

- Understanding synchronization and its role in preventing data races and ensuring thread safety.

- Synchronized methods and blocks for controlling access to shared resources.
- Using the synchronized keyword and locks for thread synchronization.

Concurrency Utilities:

- Introduction to Java concurrency utilities in java.util.concurrent package.
- Using Executors and ThreadPoolExecutor for managing thread pools.
- Implementing concurrent tasks with Callable and Future interfaces.

Thread Communication and Coordination:

- Inter-thread communication using wait(), notify(), and notifyAll() methods.
- Implementing producer-consumer and reader-writer scenarios.
- Using concurrent collections for thread-safe data structures.

Concurrency Challenges and Best Practices:

- Identifying common concurrency issues: deadlock, livelock, and race conditions.
- Best practices for writing concurrent and thread-safe code.
- Debugging and troubleshooting concurrency-related problems.

Learning Objectives:

- Understand the principles of multithreading and concurrency in Java.
- Gain proficiency in creating and managing threads, synchronizing access to shared resources, and coordinating thread execution.
- Learn about Java concurrency utilities and best practices for writing concurrent and thread-safe code.

Instructional Methods:

- Lectures: Detailed explanations of multithreading and concurrency concepts, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' multithreading implementations and provide feedback.
- Case Studies: Analysis of real-world scenarios where multithreading and concurrency are used to solve programming problems.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in multithreading and concurrency concepts.
- Quiz and Tests: Regular quizzes and tests to assess understanding of multithreading and concurrency principles.
- Project Work: Evaluation of project work to assess practical application of multithreading and concurrency in Java programming.

8. **Module 8: JDBC and Database Connectivity**

- Introduction to JDBC (Java Database Connectivity)
- Establishing database connections using JDBC
- Executing SQL queries and updating data
- Handling transactions and managing resources in JDBC

Key Topics:

Introduction to JDBC:
- Understanding the role of JDBC in database connectivity for Java applications.
- Overview of JDBC architecture: DriverManager, Connection, Statement, ResultSet.
- Benefits and advantages of using JDBC for database interaction.

Establishing Database Connections:
- Loading JDBC drivers and registering them with DriverManager.
- Establishing database connections using DriverManager.getConnection().
- Configuring database connection properties: URL, username, password.

Executing SQL Queries:
- Creating and executing SQL statements: Statement and PreparedStatement.
- Performing CRUD (Create, Read, Update, Delete) operations with JDBC.
- Handling result sets and processing query results using ResultSet.

Transaction Management:
- Understanding transactions and their properties (ACID).
- Managing transactions in JDBC: commit, rollback, and savepoints.
- Ensuring data consistency and integrity in database operations.

Handling Database Exceptions:

- Handling database-related exceptions in JDBC.
- Strategies for error handling and recovery in database operations.
- Best practices for handling database resources and closing connections.

Learning Objectives:

- Understand the principles of Java Database Connectivity (JDBC) and its role in database interaction.
- Gain proficiency in establishing database connections, executing SQL queries, and managing transactions using JDBC.
- Learn how to handle database exceptions and ensure data consistency in JDBC applications.

Instructional Methods:

- Lectures: Detailed explanations of JDBC concepts and techniques, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' JDBC implementations and provide feedback.
- Case Studies: Analysis of real-world scenarios where JDBC is used to interact with databases in Java applications.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in using JDBC to perform database operations.
- Quiz and Tests: Regular quizzes and tests to assess understanding of JDBC concepts and techniques.
- Project Work: Evaluation of project work to assess practical application of JDBC in Java programming.

9. **Module 9: Unit Testing with JUnit**

- Introduction to unit testing and test-driven development (TDD)
- Writing and executing unit tests with JUnit framework
- Test fixtures, assertions, and annotations in JUnit
- Test suites and parameterized tests

Key Topics:

Introduction to Unit Testing:
- Understanding the importance of unit testing in software development.
- Overview of unit testing principles: test cases, assertions, and test suites.
- Benefits of automated unit testing and its role in continuous integration.

Getting Started with JUnit:
- Introduction to JUnit framework and its features.
- Setting up JUnit in Java projects using build tools like Maven or Gradle.
- Writing and executing simple JUnit test cases.

Writing Test Cases with JUnit:
- Defining test methods using @Test annotation.
- Organizing test cases into test suites using @RunWith and @Suite annotations.
- Using assertions to verify expected outcomes in test cases.

Test Fixtures and Setup/Teardown:
- Understanding test fixtures and their role in setting up test environments.
- Using @Before and @After annotations for setup and teardown operations.
- Managing shared resources and dependencies in test cases.

Parameterized Tests and Test Suites:
- Writing parameterized tests using @Parameterized annotation.
- Creating test suites to organize and execute multiple test classes.
- Running tests in parallel and configuring test execution order.

Learning Objectives:

- Understand the principles of unit testing and its importance in software development.
- Gain proficiency in writing and executing unit tests using the JUnit framework.

- Learn how to organize test cases, manage test fixtures, and write parameterized tests with JUnit.

Instructional Methods:

- Lectures: Detailed explanations of unit testing concepts and JUnit framework features, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' unit test implementations and provide feedback.
- Case Studies: Analysis of real-world scenarios where unit testing is used to ensure code quality and reliability.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in writing unit tests with JUnit.
- Quiz and Tests: Regular quizzes and tests to assess understanding of unit testing principles and JUnit features.
- Project Work: Evaluation of project work to assess practical application of unit testing with JUnit in Java programming.

10. Module 10: Building RESTful APIs with Java

- Overview of REST architecture and principles
- Creating RESTful services with JAX-RS (Java API for RESTful Web Services)
- Handling HTTP methods, request/response formats, and error handling
- Documenting and testing RESTful APIs

Key Topics:

Introduction to RESTful APIs:

- Understanding the principles of REST (Representational State Transfer) architecture.

- Overview of RESTful API design principles: resources, URIs, HTTP methods, and status codes.
- Benefits of RESTful APIs and their role in modern web development.

Choosing a Java Framework:

- Overview of popular Java frameworks for building RESTful APIs: Spring Boot, JAX-RS (Java API for RESTful Web Services), and Dropwizard.
- Understanding the features, strengths, and use cases of each framework.
- Selecting the appropriate framework based on project requirements and preferences.

Designing RESTful APIs:

- Designing resource representations, URIs, and request/response formats.
- Choosing appropriate HTTP methods (GET, POST, PUT, DELETE) for CRUD operations.
- Handling query parameters, request headers, and request/response formats.

Implementing RESTful Services:

- Creating RESTful endpoints using annotations or configuration.
- Implementing CRUD operations for managing resources.
- Validating input data and handling errors gracefully.

Documenting and Testing APIs:

- Documenting RESTful APIs using Swagger or OpenAPI Specification (OAS).
- Testing APIs using tools like Postman or REST Assured.
- Writing unit tests and integration tests for RESTful services.

Learning Objectives:

- Understand the principles of RESTful architecture and API design.
- Gain proficiency in building RESTful APIs using popular Java frameworks.
- Learn how to design, implement, document, and test RESTful services in Java.

Instructional Methods:

- Lectures: Detailed explanations of RESTful architecture, API design principles, and Java frameworks, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.

- Code Reviews: Peer or instructor-led code reviews to evaluate participants' API implementations and provide feedback.
- Case Studies: Analysis of real-world RESTful API implementations and best practices.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in designing, implementing, documenting, and testing RESTful APIs in Java.
- Quiz and Tests: Regular quizzes and tests to assess understanding of RESTful architecture, API design principles, and Java frameworks.
- Project Work: Evaluation of project work to assess practical application of building RESTful APIs with Java.

11. Module 11: Introduction to Spring Framework

- Overview of the Spring Framework and its features
- Dependency Injection and Inversion of Control (IoC)
- Configuring Spring beans and application context
- Integrating Spring with Java applications

Key Topics:

Introduction to Spring Framework:
- Understanding the history and evolution of the Spring Framework.
- Overview of the core features and modules of Spring.

Dependency Injection (DI):
- Understanding the concept of dependency injection and its benefits.
- Configuring beans and dependencies using XML-based and annotation-based configurations.
- Implementing inversion of control (IoC) using Spring's DI container.

Aspect-Oriented Programming (AOP):
- Understanding the principles of aspect-oriented programming (AOP).
- Creating aspects and pointcuts to implement cross-cutting concerns.

- Using AOP features such as advice, joinpoints, and weaving in Spring applications.

Declarative Transaction Management:

- Understanding transaction management and its importance in enterprise applications.
- Configuring declarative transaction management using Spring's transaction management support.
- Using transactional annotations to define transactional behavior.

Spring Data Access:

- Overview of Spring's support for data access technologies such as JDBC, Hibernate, and JPA (Java Persistence API).
- Using Spring templates and repositories for simplified data access.
- Implementing DAO (Data Access Object) patterns with Spring.

Learning Objectives:

- Understand the core features and modules of the Spring Framework.
- Gain proficiency in dependency injection, aspect-oriented programming, and declarative transaction management with Spring.
- Learn how to use Spring for simplified data access and integration with data access technologies.

Instructional Methods:

- Lectures: Detailed explanations of Spring Framework concepts and features, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' Spring configurations and implementations.
- Case Studies: Analysis of real-world scenarios where Spring Framework is used to develop enterprise applications.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in configuring and using Spring features.
- Quiz and Tests: Regular quizzes and tests to assess understanding of Spring Framework concepts and features.
- Project Work: Evaluation of project work to assess practical application of Spring Framework in Java enterprise development.

12. Module 1: Introduction to Java 8

- Overview of Java 8 features
- Introduction to lambda expressions
- Stream API basics

Key Topics:

Introduction to Java 8:

- Overview of the Java 8 release and its significance in the evolution of the Java platform.
- Understanding the goals and objectives of Java 8.
- Comparison with previous Java versions and key differences.

Lambda Expressions:

- Introduction to lambda expressions and their syntax.
- Understanding the benefits of lambda expressions for writing concise and expressive code.
- Examples demonstrating the use of lambda expressions in various contexts.

Stream API:

- Introduction to the Stream API for processing collections in a functional style.
- Overview of intermediate and terminal operations provided by the Stream API.
- Examples illustrating the use of streams for data processing tasks.

Learning Objectives:

- Understand the motivations behind the introduction of Java 8 and its key features.
- Gain proficiency in using lambda expressions to write concise and expressive code.
- Learn how to leverage the Stream API for efficient data processing tasks.

Instructional Methods:

- Lectures: Detailed explanations of Java 8 features and enhancements, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' understanding and implementation of Java 8 concepts.
- Case Studies: Analysis of real-world scenarios where Java 8 features are applied to solve programming problems.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in using lambda expressions and the Stream API.
- Quiz and Tests: Regular quizzes and tests to assess understanding of Java 8 concepts and features.
- Project Work: Evaluation of project work to assess practical application of Java 8 features in Java programming.

12.1. Module 2: Lambda Expressions and Functional Interfaces

- Understanding lambda expressions syntax
- Functional interfaces and their role
- Method references and constructor references

Key Topics:

Lambda Expressions Syntax:

- Understanding the syntax of lambda expressions: parameter list, arrow token, and body.
- Writing lambda expressions for different types of functional interfaces.
- Examples demonstrating lambda expressions in action.

Functional Interfaces:

- Introduction to functional interfaces and their role in lambda expressions.
- Defining custom functional interfaces using the @FunctionalInterface annotation.
- Overview of common functional interfaces in the java.util.function package.

Method References:

- Understanding method references and their relationship with lambda expressions.
- Different types of method references: static, instance, and constructor references.
- Examples illustrating the use of method references in various scenarios.

Learning Objectives:

- Understand the syntax and usage of lambda expressions in Java 8.
- Gain proficiency in defining and using functional interfaces.
- Learn how to leverage method references to write more concise and readable code.

Instructional Methods:

- Lectures: Detailed explanations of lambda expressions syntax, functional interfaces, and method references, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' understanding and implementation of lambda expressions and functional interfaces.
- Case Studies: Analysis of real-world scenarios where lambda expressions and functional interfaces are used to solve programming problems.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in using lambda expressions, functional interfaces, and method references.
- Quiz and Tests: Regular quizzes and tests to assess understanding of lambda expressions syntax and usage.
- Project Work: Evaluation of project work to assess practical application of lambda expressions and functional interfaces in Java programming.

12.2. Module 3: Stream API

- Introduction to Stream API
- Stream operations: map, filter, reduce.
- Stream collectors and terminal operations

Key Topics:

Introduction to Stream API:
- Understanding the purpose and benefits of the Stream API.
- Overview of the Stream interface and its characteristics.
- Comparison between streams and traditional collection manipulation.

Stream Operations:
- Exploring intermediate stream operations: map, filter, sorted, distinct, etc.
- Understanding terminal stream operations: forEach, collect, reduce.
- Chaining multiple stream operations for complex data processing tasks.

Parallel Streams:
- Introduction to parallel streams and their benefits for concurrent processing.
- Using parallel streams to leverage multi-core processors for improved performance.
- Considerations and best practices for using parallel streams effectively.

Stream Collectors:
- Understanding stream collectors and their role in converting streams to collections or other data structures.
- Exploring common collectors: toList, toSet, joining, groupingBy, partitioningBy.

- Writing custom collectors for specialized data processing requirements.

Learning Objectives:

- Understand the purpose and benefits of the Stream API in Java 8.
- Gain proficiency in using stream operations to perform data processing tasks on collections.
- Learn how to leverage parallel streams and stream collectors for efficient and flexible data processing.

Instructional Methods:

- Lectures: Detailed explanations of Stream API concepts and operations, accompanied by examples and demonstrations.
- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' understanding and implementation of stream operations.
- Case Studies: Analysis of real-world scenarios where the Stream API is used to solve data processing problems.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in using stream operations and collectors.
- Quiz and Tests: Regular quizzes and tests to assess understanding of Stream API concepts and usage.
- Project Work: Evaluation of project work to assess practical application of the Stream API in Java programming.

12.3. Module 4: Optional Class

- Understanding the Optional class
- Dealing with null values effectively

- Best practices for using Optional.

Key Topics:

Introduction to Optional Class:
- Understanding the motivation behind the Optional class.
- Overview of the Optional class and its methods.
- Comparison between using Optional and traditional null checks.

Creating Optional Instances:
- Creating Optional instances using static factory methods: of, ofNullable, empty.
- Understanding the difference between of and ofNullable methods.
- Handling null values and absent values using Optional.

Using Optional Methods:
- Exploring methods provided by the Optional class: get, isPresent, ifPresent, orElse, orElseGet, orElseThrow.
- Chaining Optional methods for concise and expressive code.
- Best practices for using Optional methods effectively.

Combining Optional with Stream API:
- Leveraging Optional in Stream API operations.
- Using methods like findFirst, findAny, max, min with Optional.
- Handling absent values gracefully in stream processing.

Learning Objectives:

- Understand the purpose and benefits of the Optional class in Java 8.
- Gain proficiency in creating Optional instances and using Optional methods effectively.
- Learn how to integrate Optional with Stream API for cleaner and safer code.

Instructional Methods:

- Lectures: Detailed explanations of Optional class concepts and methods, accompanied by examples and demonstrations.

- Hands-on Practice: Interactive coding exercises and assignments to reinforce learning through practical application.
- Code Reviews: Peer or instructor-led code reviews to evaluate participants' understanding and implementation of Optional usage.
- Case Studies: Analysis of real-world scenarios where Optional class is used to handle null values effectively.

Assessment:

- Coding Assignments: Completion of coding assignments to demonstrate proficiency in using Optional class and integrating it with Stream API.
- Quiz and Tests: Regular quizzes and tests to assess understanding of Optional class concepts and methods.
- Project Work: Evaluation of project work to assess practical application of Optional class in Java programming.

12.4. Module 5: Default and Static Methods

- Introduction to default methods in interfaces
- Static methods in interfaces
- Use cases and best practices.

12.5. Module 6: Date and Time API

- Overview of Date and Time API in Java 8
- LocalDate, LocalTime, LocalDateTime
- Formatting and parsing dates and times

12.6. Module 7: Completable Future

- Introduction to Completable Future
- Asynchronous programming in Java 8
- Combining and chaining Completable Future

12.7. Module 8: Method References and Constructor References

- Understanding method references
- Constructor references and their usage
- Best practices and examples

12.8. Module 9: New Features in Collections

- Overview of new features in Collections API
- Introduction to new methods in List, Set, Map
- Use cases and examples.

12.9. Module 10: Concurrency Enhancements

- Introduction to concurrency enhancements in Java 8
- CompletableFuture for asynchronous programming
- Atomic variables and non-blocking algorithms

12.10. Module 11: Nashorn JavaScript Engine

- Introduction to Nashorn JavaScript Engine
- Interoperability between Java and JavaScript
- Use cases and examples.

12.11. Module 12: Parallel Streams

- Understanding parallel streams
- Parallel processing with streams
- Performance considerations and best practices

12.12. Module 13: Default Methods in Interfaces

- Deep dive into default methods in interfaces
- Multiple inheritance with default methods
- Best practices and examples

12.13. Module 14: API Updates and Enhancements

- Overview of various API updates and enhancements in Java 8
- Updates in java.util, java.util.concurrent, java.util.function packages
- Use cases and examples.

12.14. Mock test on concepts
12.15. Free interview skills
12.16. Live project