Especialización Java Standard Edition

Fundamentos del Lenguaje Java

Nivel: Básico / Iniciación | Código: UNW-JSE-101 | Duración: 20 horas académicas

uneweb.edu.ve
Programa de Formación en Desarrollo de Software

Java SE — Nivel I

Fundamentos del Lenguaje Java

Presentación del curso

Este curso introduce al participante en el lenguaje Java desde cero. Java es uno de los lenguajes más utilizados en el mundo, presente en aplicaciones empresariales, Android, servicios web y sistemas embebidos. Al finalizar, el estudiante dominará la sintaxis básica, la programación orientada a objetos y podrá construir programas funcionales ejecutables desde la línea de comandos o un IDE.

Objetivo general

Capacitar al participante en los fundamentos del lenguaje Java SE: sintaxis, tipos de datos, estructuras de control, métodos y los pilares de la programación orientada a objetos (POO), para que pueda desarrollar programas básicos y sentar las bases del nivel II.

Objetivos específicos

- Instalar y configurar el JDK y un IDE (IntelliJ IDEA o VS Code) para el desarrollo en Java.
- Comprender la arquitectura Java: JVM, JRE, JDK y el proceso de compilación/ejecución.
- Dominar la sintaxis básica: variables, tipos de datos primitivos, operadores y conversiones de tipo.
- Controlar el flujo de un programa con estructuras condicionales (if, switch) y ciclos (for, while, do-while).
- Definir y llamar métodos; comprender el paso por valor y la sobrecarga de métodos.
- Modelar entidades del mundo real mediante clases y objetos.
- Aplicar los principios de encapsulamiento y herencia.
- Utilizar arreglos unidimensionales y multidimensionales, y la clase Arrays.

Dirigido a

- Personas sin experiencia previa en programación que deseen aprender Java desde cero.
- Profesionales de otras áreas que quieran ingresar al desarrollo de software.
- Estudiantes que deseen construir una base sólida en un lenguaje de uso empresarial.

Requisitos previos

- Conocimientos básicos de computación (uso de archivos, navegación web).
- Computador con Windows, macOS o Linux y 8 GB de RAM recomendados.

Metodología

Cada módulo alterna exposición conceptual con laboratorios de código guiados. El participante escribe y ejecuta código desde la primera sesión. Claude.ai se utiliza como asistente para depurar errores, explicar mensajes del compilador y generar ejemplos alternativos que refuercen cada concepto.

Plan de contenidos (20 horas académicas)

Módulo	Tema	Horas
1	Introducción a Java: historia, JVM, JDK, instalación de IntelliJ IDEA y primer programa «Hola Mundo»	2
2	Tipos de datos primitivos, variables, constantes, operadores y conversiones de tipo (casting)	2
3	Estructuras condicionales: if, if-else, if-else if, switch y operador ternario	2
4	Ciclos: for, while, do-while, break, continue y ciclos anidados	2
5	Métodos: declaración, retorno, parámetros, sobrecarga y paso por valor	3
6	Programación Orientada a Objetos I: clases, objetos, atributos, constructores y palabra clave this	3
7	Encapsulamiento: modificadores de acceso, getters/setters; Herencia: extends, super y polimorfismo de subtipo	3
8	Arreglos: unidimensionales, multidimensionales, clase Arrays y proyecto final integrador	3

Detalle de módulos

Módulo 1 — Introducción a Java y configuración del entorno

- Historia de Java: origen en Sun Microsystems (1995), adquisición por Oracle y versiones LTS (8, 11, 17, 21).
- Arquitectura: JDK (compilador javac), JRE (Java Runtime Environment) y JVM (Java Virtual Machine).
- Write Once Run Anywhere: bytecode y portabilidad entre plataformas.
- Instalación del JDK 21 LTS y configuración de JAVA_HOME.
- Primeros pasos en IntelliJ IDEA Community: creación de proyecto, estructura de carpetas y ejecución.
- Anatomía de un programa Java: clase pública, método main, sentencia System.out.println.

Módulo 2 — Tipos de datos, variables y operadores

- Tipos primitivos: byte, short, int, long, float, double, boolean, char.
- Declaración, inicialización y convenciones de nomenclatura (camelCase).
- Constantes con final y convención SNAKE_UPPER_CASE.
- Operadores: aritméticos, de asignación, relacionales, lógicos (&&, ||, !) y bit a bit básicos.
- Casting implícito (widening) y explícito (narrowing); conversión con Integer.parseInt, Double.parseDouble.
- Entrada de datos por consola con Scanner.

Módulo 3 — Estructuras condicionales

- if simple, if-else y cadenas if-else if-else para múltiples condiciones.
- Operador ternario (condición ? valorTrue : valorFalse).
- switch clásico con break y fall-through; switch expression (Java 14+) con ->.
- Buenas prácticas: evitar condiciones negadas innecesarias, default en switch.
- Laboratorio: calculadora de calificaciones con categorías (Excelente, Aprobado, Reprobado).

Módulo 4 — Ciclos e iteración

- Ciclo for: inicialización, condición, actualización; for-each sobre arreglos.
- Ciclo while: condición de entrada; do-while: condición de salida (ejecuta al menos una vez).
- Sentencias break (salida inmediata) y continue (saltar iteración actual).
- Ciclos anidados: matrices, tablas y patrones.

- Laboratorio: tabla de multiplicar, números primos entre 1 y 100 y serie de Fibonacci.

Módulo 5 — Métodos

- Declaración: modificador de acceso, tipo de retorno, nombre y lista de parámetros.
- Sentencia return; métodos void; convención de nomenclatura verbAction.
- Paso de argumentos por valor (primitivos) vs. por referencia (objetos).
- Sobrecarga de métodos (method overloading): mismo nombre, distinta firma.
- Recursividad: concepto, caso base y caso recursivo; ejemplos con factorial y Fibonacci.
- Laboratorio: biblioteca de métodos matemáticos (potencia, factorial, MCD con algoritmo de Euclides).

Módulo 6 — POO I: Clases y objetos

- ¿Qué es la POO? Los cuatro pilares: encapsulamiento, herencia, polimorfismo y abstracción.
- Declaración de clase: atributos (campos) y métodos de instancia.
- Constructores: sin argumentos y con parámetros; constructor por defecto.
- Palabra clave this: referencia al objeto actual y resolución de ambigüedad.
- Instanciación con new; memoria en el heap y referencias en el stack.
- Laboratorio: clase Cuenta bancaria con atributos titular, saldo y métodos depositar, retirar, mostrarSaldo.

Módulo 7 — Encapsulamiento y herencia

- Modificadores de acceso: private, package-private (default), protected, public.
- Getters y setters: convención JavaBeans y validación de datos en setters.
- Herencia con extends: reutilización de código, relación «es un».
- Palabra clave super: llamada al constructor padre y a métodos sobrescritos.
- Sobreescritura de métodos (@Override) y polimorfismo de subtipo.
- Laboratorio: jerarquía Vehiculo → Automovil, Moto con atributos y comportamientos propios.

Módulo 8 — Arreglos y proyecto final

- Arreglos unidimensionales: declaración, inicialización, acceso por índice y length.
- Recorrido con for y for-each; búsqueda lineal y ordenamiento con Arrays.sort.
- Arreglos bidimensionales: matrices, recorrido y casos de uso.
- Clase java.util.Arrays: toString, copyOf, fill, binarySearch.
- Proyecto final integrador: sistema de registro de estudiantes con arreglo de objetos Estudiante, menú de opciones (agregar, listar, buscar, calcular promedio).

Evaluación

- Ejercicios prácticos por módulo (40%).
- Quiz de sintaxis y POO básica (20%).
- Proyecto final integrador (40%).

Recursos y herramientas

- JDK 21 LTS (OpenJDK o Oracle JDK) — gratuito.
- IntelliJ IDEA Community Edition o VS Code con extensión Java Extension Pack.
- Claude.ai para depuración de errores del compilador y generación de ejemplos alternativos.

Especialización Java Standard Edition

POO Avanzada, Colecciones y Manejo de Excepciones

Nivel: Intermedio | Código: UNW-JSE-201 | Duración: 20 horas académicas

uneweb.edu.ve
Programa de Formación en Desarrollo de Software

Java SE — Nivel II

POO Avanzada, Colecciones y Manejo de Excepciones

Presentación del curso

El Nivel II profundiza en los pilares avanzados de la Programación Orientada a Objetos en Java: interfaces, clases abstractas, polimorfismo profundo, el Framework de Colecciones (Collections Framework), manejo robusto de excepciones, trabajo con cadenas de texto, fechas y el sistema de entrada/salida básico. El participante estará preparado para construir aplicaciones Java bien estructuradas y mantenibles.

Objetivo general

Desarrollar en el participante la capacidad de diseñar jerarquías de clases complejas, utilizar las estructuras de datos del Collections Framework, manejar errores de forma robusta y operar con los tipos de la API estándar de Java SE más utilizados en la práctica profesional.

Objetivos específicos

- Definir y utilizar clases abstractas e interfaces para diseño por contrato.
- Aplicar polimorfismo avanzado: upcast, downcast e instanceof (pattern matching Java 16+).
- Utilizar las interfaces funcionales y el patrón de diseño Iterator.
- Manejar las colecciones más importantes: ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap.
- Diseñar jerarquías de excepciones propias y manejar errores con try-catch-finally y multi-catch.
- Manipular cadenas con String, StringBuilder y expresiones regulares básicas.
- Trabajar con fechas y tiempos usando la API java.time (LocalDate, LocalDateTime, Duration).
- Leer y escribir archivos de texto con java.io y java.nio.file.

Requisitos previos

- Haber aprobado Java SE Nivel I (UNW-JSE-101) o dominar POO básica en Java.
- Comodidad con el uso del IDE y la compilación/ejecución de programas Java.

Metodología

El hilo conductor del curso es un sistema de gestión de una tienda de productos tecnológicos que evoluciona módulo a módulo. Cada sesión agrega una capa de complejidad al sistema, conectando los conceptos teóricos con necesidades reales de negocio. Claude.ai se usa para revisar diseños de clases, sugerir alternativas de implementación y comparar estructuras de datos.

Plan de contenidos (20 horas académicas)

Módulo	Tema	Horas
1	Clases abstractas e interfaces: diferencias, contratos y diseño por abstracción	3
2	Polimorfismo avanzado: upcast, downcast, instanceof con pattern matching, Comparable y Comparator	2
3	Collections Framework I: List (ArrayList, LinkedList), Set (HashSet, TreeSet, LinkedHashSet)	3
4	Collections Framework II: Map (HashMap, TreeMap, LinkedHashMap); iteración con entrySet y Stream básico	2
5	Manejo de excepciones: checked vs unchecked, try-catch-finally, multi-catch, try-with-resources y excepciones propia	3
6	String avanzado: métodos de String, StringBuilder/StringBuffer, String.format y expresiones regulares con Pattern/Ma	2
7	API java.time: LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Duration, Period y DateTimeFormatter	2
8	E/S básica: lectura/escritura de archivos con Files, Path, BufferedReader/Writer y proyecto final	3

Detalle de módulos

Módulo 1 — Clases abstractas e interfaces

- Clase abstracta: métodos abstractos y concretos; no se puede instanciar directamente.
- Interface: contrato puro; métodos default y static (Java 8+); métodos privados (Java 9+).
- Diferencias clave: herencia simple (extends) vs. implementación múltiple de interfaces (implements).
- Principio de Segregación de Interfaces (ISP) del SOLID: interfaces pequeñas y cohesivas.
- Laboratorio: jerarquía Producto (abstracta) → ProductoFisico, ProductoDigital con interface Descargable.

Módulo 2 — Polimorfismo avanzado y ordenación

- Upcast automático y downcast explícito; ClassCastException y cómo evitarla.
- Operador instanceof clásico y pattern matching: if (obj instanceof Tipo t) { ... }
- Interface Comparable: método compareTo para ordenación natural.
- Interface Comparator: ordenación alternativa con lambda y Comparator.comparing().
- Collections.sort y List.sort con comparadores; orden inverso con reversed().

Módulo 3 — Collections Framework I: List y Set

- ArrayList: acceso O(1), inserción O(n); operaciones add, remove, get, set, subList.
- LinkedList: acceso O(n), inserción O(1); uso como Deque (addFirst, addLast, poll).
- HashSet: sin duplicados, sin orden, O(1) promedio; importancia de equals/hashCode.
- TreeSet: sin duplicados, orden natural o comparador, O(log n); NavigableSet.
- LinkedHashSet: sin duplicados, orden de inserción.
- Laboratorio: catálogo de productos sin duplicados (HashSet) y listado ordenado por precio (TreeSet con Comparator).

Módulo 4 — Collections Framework II: Map e introducción a Stream

- HashMap: pares clave-valor, O(1) promedio; métodos put, get, containsKey, getOrDefault, putIfAbsent.
- TreeMap: claves en orden natural; NavigableMap, floorKey, ceilingKey.
- LinkedHashMap: orden de inserción; uso como caché LRU básico.
- Iteración: entrySet(), keySet(), forEach con lambda.

- Introducción a Stream API: stream(), filter(), map(), collect(Collectors.toList()), count().

Módulo 5 — Manejo de excepciones

- Jerarquía: Throwable → Error / Exception → RuntimeException (unchecked) / checked exceptions.
- try-catch-finally: orden de ejecución y el bloque finally garantizado.
- Multi-catch: catch (IOException | SQLException e) para reducir duplicación.
- try-with-resources: AutoCloseable y cierre automático de recursos.
- Creación de excepciones propias: extends Exception o extends RuntimeException; constructores con mensaje y causa.
- Buenas prácticas: no swallow exceptions, loguear la causa raíz, lanzar excepciones significativas.

Módulo 6 — String avanzado y expresiones regulares

- Inmutabilidad de String: pool de strings y operador == vs. equals().
- Métodos esenciales: substring, indexOf, contains, startsWith, endsWith, replace, split, strip, repeat.
- StringBuilder: append, insert, delete, reverse; cuándo preferirlo sobre concatenación.
- String.format y formatted() (Java 15+): especificadores %s, %d, %f, %n.
- Expresiones regulares con Pattern y Matcher: validación de emails, teléfonos y RIF.

Módulo 7 — API java.time

- Motivación: problemas de java.util.Date y Calendar; inmutabilidad en java.time.
- LocalDate, LocalTime, LocalDateTime: creación con of(), now(), parse().
- ZonedDateTime y ZoneId: manejo de zonas horarias.
- Duration (tiempo entre instantes) y Period (diferencia en días/meses/años).
- DateTimeFormatter: formateo personalizado y localización con Locale.
- Laboratorio: cálculo de antigüedad de empleados, agenda de eventos con zonas horarias.

Módulo 8 — Entrada/Salida y proyecto final

- java.nio.file.Path y Files: createFile, createDirectory, readAllLines, writeString, copy, delete.
- Lectura eficiente con BufferedReader y Files.lines() (Stream).
- Escritura con BufferedWriter y PrintWriter.
- Serialización básica: Serializable, ObjectOutputStream y ObjectInputStream.
- Proyecto final: sistema de tienda tecnológica con catálogo en colecciones, excepciones propias, persistencia en archivo de texto y menú interactivo.

Evaluación

- Laboratorios prácticos por módulo (40%).
- Quiz de colecciones, excepciones y API java.time (20%).
- Proyecto final sistema de tienda (40%).

Recursos y herramientas

- JDK 21 LTS, IntelliJ IDEA Community.
- Documentación oficial de Java SE (docs.oracle.com/en/java/javase/21/).
- Claude.ai para diseño de jerarquías de clases, revisión de código y comparación de estructuras de datos.

Especialización Java Standard Edition

Programación Funcional, Genéricos y Concurrencia

Nivel: Avanzado | Código: UNW-JSE-301 | Duración: 20 horas académicas

uneweb.edu.ve
Programa de Formación en Desarrollo de Software

Java SE — Nivel III

Programación Funcional, Genéricos y Concurrencia

Presentación del curso

El Nivel III lleva al participante a las características avanzadas de Java SE moderno: programación funcional con Lambdas y Stream API, genéricos para código reutilizable y seguro en tipos, programación concurrente con hilos y el framework java.util.concurrent, Optional para el manejo explícito de nulos, y las últimas características del lenguaje (Java 14-21): records, sealed classes, text blocks y switch expressions.

Objetivo general

Capacitar al participante para escribir código Java moderno, idiomático y eficiente utilizando el paradigma funcional, tipos genéricos, concurrencia segura y las últimas características del lenguaje, habilitándolo para trabajar en proyectos empresariales de nivel profesional.

Objetivos específicos

- Crear y utilizar expresiones lambda e interfaces funcionales (Function, Predicate, Consumer, Supplier).
- Construir pipelines de procesamiento de datos con Stream API: filter, map, flatMap, reduce, collect.
- Diseñar clases, interfaces y métodos genéricos con comodines (wildcards) y bounds.
- Manejar la ausencia de valores de forma explícita con Optional.
- Crear y gestionar hilos con Thread, Runnable y el ExecutorService.
- Utilizar estructuras de datos thread-safe (ConcurrentHashMap, BlockingQueue) y sincronización.
- Aplicar CompletableFuture para programación asíncrona no bloqueante.
- Utilizar records, sealed classes, text blocks y switch expressions de Java moderno.

Requisitos previos

- Haber aprobado Java SE Nivel II (UNW-JSE-201).
- Sólido manejo de POO, colecciones y manejo de excepciones en Java.

Metodología

El caso de uso central es un sistema de procesamiento de datos de ventas (carga de CSV, transformación funcional, agrupación y generación de reportes), al que se añade un módulo de descarga paralela de precios desde múltiples fuentes. Claude.ai se emplea para explicar el comportamiento del Stream API, revisar el diseño de tipos genéricos y analizar condiciones de carrera en código concurrente.

Plan de contenidos (20 horas académicas)

Módulo	Tema	Horas
1	Lambdas e interfaces funcionales: sintaxis, contexto de captura y las 4 interfaces funcionales clave	3
2	Stream API I: creación de streams, operaciones intermedias (filter, map, flatMap, distinct, sorted, limit)	3
3	Stream API II: operaciones terminales (collect, reduce, count, min, max, anyMatch, findFirst) y Collectors avanzados	3
4	Genéricos: clases e interfaces genéricas, métodos genéricos, wildcards (?, ? extends, ? super) y type erasure	2
5	Optional<T>: creación, transformación (map, flatMap), consumo (ifPresent, orElse, orElseThrow) y antipatrones	1
6	Concurrencia I: Thread, Runnable, Callable, ciclo de vida, sincronización con synchronized y volatile	3
7	Concurrencia II: ExecutorService, Future, CompletableFuture, estructuras thread-safe y problemas clásicos (deadlock	3
8	Java moderno (14-21): records, sealed classes, text blocks, switch expression y proyecto final de pipeline funcional	2

Detalle de módulos

Módulo 1 — Lambdas e interfaces funcionales

- ¿Qué es una lambda? Sintaxis: (params) -> expresión y (params) -> { bloque }.
- Interfaces funcionales (@FunctionalInterface): exactamente un método abstracto.
- Function: transformación; Predicate: filtrado booleano; Consumer: efecto sin retorno; Supplier: generación.
- BiFunction, UnaryOperator, BinaryOperator y sus formas primitivas (IntFunction, etc.).
- Referencias de método: Clase::metodoEstatico, objeto::metodoInstancia, Clase::new.
- Captura de variables: efectivamente final; closure en Java.

Módulo 2 — Stream API I: operaciones intermedias

- Creación de streams: Collection.stream(), Arrays.stream(), Stream.of(), Stream.iterate(), Stream.generate(), Files.lines().
- Naturaleza lazy de las operaciones intermedias: evaluación diferida hasta la operación terminal.
- filter(Predicate): selección de elementos; map(Function): transformación de tipo.
- flatMap(Function>): aplanamiento de streams anidados.
- distinct(), sorted(), sorted(Comparator), limit(n), skip(n), peek().
- Laboratorio: pipeline de filtrado y transformación de un catálogo de productos cargado desde CSV.

Módulo 3 — Stream API II: operaciones terminales y Collectors

- Operaciones terminales: forEach, count, min, max, findFirst, findAny, anyMatch, allMatch, noneMatch.
- reduce(identity, BinaryOperator): acumulación manual; Optional como resultado.
- collect(Collector): el recolector más poderoso.
- Collectors: toList, toSet, toMap, joining, groupingBy, partitioningBy, counting, summarizingInt.
- Collectors.groupingBy anidado: agrupación en dos niveles.
- Streams de primitivos: IntStream, LongStream, DoubleStream; range, rangeClosed, sum, average, stats.

Módulo 4 — Genéricos

- Motivación: seguridad de tipos en tiempo de compilación vs. Object con casting.
- Clase genérica: class Caja { T valor; }; instanciación y diamond operator (<>).

- Interface genérica y su implementación; múltiples parámetros de tipo.
- Método genérico: T primerElemento(List lista).
- Wildcards: ? (unbounded), ? extends T (upper bounded, covariante), ? super T (lower bounded, contravariante).
- Type erasure: cómo el compilador elimina tipos genéricos en bytecode y sus implicaciones.

Módulo 5 — Optional

- El problema del NullPointerException: el «billion dollar mistake».
- Creación: Optional.of(valor), Optional.ofNullable(valor), Optional.empty().
- Comprobación: isPresent(), isEmpty() (Java 11+).
- Transformación funcional: map(), flatMap() para Optionals anidados.
- Consumo: ifPresent(Consumer), ifPresentOrElse() (Java 9+), orElse(T), orElseGet(Supplier), orElseThrow(Supplier).
- Antipatrones: Optional como campo de clase, Optional en parámetros de métodos.

Módulo 6 — Concurrencia I: hilos y sincronización

- Proceso vs. hilo; ventajas del multithreading: paralelismo y concurrencia.
- Crear hilos: extender Thread o implementar Runnable; método start() vs. run().
- Ciclo de vida: NEW → RUNNABLE → BLOCKED/WAITING/TIMED_WAITING → TERMINATED.
- Condición de carrera (race condition): acceso no sincronizado a estado compartido.
- synchronized en métodos y bloques; intrínsic lock (monitor).
- volatile: visibilidad de cambios entre hilos sin atomicidad; cuándo es suficiente.

Módulo 7 — Concurrencia II: ExecutorService y CompletableFuture

- ExecutorService: Executors.newFixedThreadPool, newCachedThreadPool, newSingleThreadExecutor.
- submit(Callable): retorna Future; get() bloqueante y get(timeout).
- ScheduledExecutorService: ejecución periódica con scheduleAtFixedRate.
- Estructuras thread-safe: ConcurrentHashMap, CopyOnWriteArrayList, ArrayBlockingQueue.
- CompletableFuture: supplyAsync, thenApply, thenAccept, thenCombine, exceptionally, allOf, anyOf.
- Deadlock: condiciones necesarias, cómo detectarlo y estrategias de prevención.

Módulo 8 — Java moderno y proyecto final

- Text blocks (Java 15): cadenas multilínea con delimitador """.
- Records (Java 16): clases inmutables de datos con constructor canónico, getters, equals/hashCode/toString automáticos.
- Sealed classes (Java 17): restricción de jerarquías; permits; compatibilidad con pattern matching.
- Switch expressions (Java 14+): -> sin fall-through, retorno de valor, exhaustividad.
- Pattern matching en switch (Java 21): case Tipo t -> ...;
- Proyecto final: pipeline de procesamiento de ventas — carga de CSV con Files.lines(), transformación con Stream API, agrupación con Collectors.groupingBy, descarga paralela de precios con CompletableFuture y reporte en archivo.

Evaluación

- Laboratorios prácticos por módulo (40%).
- Quiz de Streams, Genéricos y Concurrencia (20%).
- Proyecto final de pipeline funcional y paralelo (40%).

Recursos y herramientas

- JDK 21 LTS — requerido para records, sealed classes y pattern matching completo.
- IntelliJ IDEA Community (soporte completo para características modernas de Java).
- Dataset de ventas en CSV para el proyecto final.
- Claude.ai para análisis de comportamiento de Streams, revisión de código concurrente y diseño de genéricos.

Especialización Java Standard Edition

Patrones de Diseño, Módulos, Testing Profesional y Reflexión

Nivel: Especialización / Profesional | Código: UNW-JSE-401 | Duración: 20 horas académicas

uneweb.edu.ve
Programa de Formación en Desarrollo de Software

Java SE — Nivel IV

Patrones de Diseño, Módulos, Testing Profesional y Reflexión

Presentación del curso

El Nivel IV cierra la especialización con los conocimientos que distinguen a un desarrollador Java profesional: patrones de diseño GoF aplicados en Java, el sistema de módulos (JPMS — Java Platform Module System), pruebas automatizadas con JUnit 5 y Mockito, programación con reflexión y anotaciones, acceso a bases de datos con JDBC, y las bases de construcción de proyectos con Maven. Al finalizar, el participante estará listo para integrarse a equipos de desarrollo empresarial y continuar hacia frameworks como Spring Boot.

Objetivo general

Formar un desarrollador Java SE completo con habilidades en diseño de software (patrones GoF), calidad de código (testing con JUnit 5/Mockito), acceso a datos (JDBC), construcción de proyectos (Maven) y extensibilidad dinámica (reflexión y anotaciones personalizadas).

Objetivos específicos

- Aplicar los patrones de diseño creacionales (Singleton, Factory Method, Builder, Prototype), estructurales (Adapter, Decorator, Proxy, Facade) y de comportamiento (Strategy, Observer, Command, Template Method) en Java.
- Diseñar aplicaciones modulares con JPMS: módulos, exports, requires, services.
- Escribir pruebas unitarias con JUnit 5: anotaciones, assertions, pruebas parametrizadas y ciclo de vida.
- Aplicar dobles de prueba con Mockito: mock, stub, verify y ArgumentCaptor.
- Conectar una aplicación Java a una base de datos relacional mediante JDBC.
- Gestionar dependencias y el ciclo de vida del proyecto con Maven.
- Crear y procesar anotaciones personalizadas en tiempo de ejecución con la API de Reflexión.
- Utilizar ServiceLoader para extensibilidad mediante el patrón Plugin.

Requisitos previos

- Haber aprobado Java SE Nivel III (UNW-JSE-301).
- Familiaridad con Stream API, genéricos y concurrencia básica en Java.

Metodología

El proyecto integrador del curso es un framework de procesamiento de reportes extensible (plugin-based), construido con Maven, probado con JUnit 5 + Mockito, persistido con JDBC y empaquetado como módulo JPMS. Cada módulo agrega una capa de arquitectura al proyecto, permitiendo ver cómo los patrones y herramientas encajan en un sistema coherente. Claude.ai se utiliza para revisar implementaciones de patrones, analizar cobertura de pruebas y sugerir refactorizaciones.

Plan de contenidos (20 horas académicas)

Módulo	Tema	Horas
1	Patrones creacionales: Singleton (thread-safe), Factory Method, Abstract Factory, Builder y Prototype	3
2	Patrones estructurales: Adapter, Decorator, Proxy (dinámico con reflexión), Facade y Composite	2
3	Patrones de comportamiento: Strategy, Observer, Command, Template Method e Iterator	3
4	JUnit 5: estructura de pruebas, assertions, @ParameterizedTest, @BeforeEach/@AfterAll y pruebas de excepciones	3
5	Mockito: mock(), stub con when/thenReturn, verificación con verify(), ArgumentCaptor y spy()	2
6	JDBC: conexión a PostgreSQL/MySQL, PreparedStatement, ResultSet, transacciones y patrón DAO	3
7	Maven: estructura de proyecto, POM, ciclo de vida, dependencias, plugins y empaquetado JAR/fat-JAR	2
8	JPMS, reflexión, anotaciones personalizadas, ServiceLoader y proyecto final integrador	2

Detalle de módulos

Módulo 1 — Patrones creacionales

- Singleton: instancia única; implementación con double-checked locking e inicialización con enum.
- Factory Method: delegación de la creación a subclases; desacoplamiento del cliente.
- Abstract Factory: familia de objetos relacionados sin especificar clases concretas.
- Builder: construcción paso a paso de objetos complejos; fluent API y Builder interno estático.
- Prototype: copia de objetos con Cloneable y clone() vs. copia defensiva.
- Laboratorio: fábrica de generadores de reportes (PDF, Excel, CSV) con Factory Method.

Módulo 2 — Patrones estructurales

- Adapter: compatibilidad entre interfaces incompatibles; adaptador de clase y de objeto.
- Decorator: agregar responsabilidades dinámicamente; diferencia con herencia.
- Proxy estático y dinámico (java.lang.reflect.Proxy): intercepción de llamadas a métodos.
- Facade: interfaz simplificada sobre un subsistema complejo.
- Composite: tratamiento uniforme de objetos individuales y compuestos; árbol de componentes.
- Laboratorio: sistema de filtros de texto con Decorator (cifrado, compresión, logging).

Módulo 3 — Patrones de comportamiento

- Strategy: familia de algoritmos intercambiables; eliminar condicionales con polimorfismo.
- Observer: notificación de eventos con java.util.Observable (legacy) e implementación propia con interfaces.
- Command: encapsular una solicitud como objeto; soporte para undo/redo.
- Template Method: esqueleto de algoritmo en clase base con pasos variables en subclases.
- Iterator: recorrido secuencial sin exponer la estructura interna; implementar Iterable.
- Laboratorio: sistema de procesamiento de reportes con Strategy (distintos formatos) y Observer (notificación de eventos de progreso).

Módulo 4 — Testing con JUnit 5

- Arquitectura de JUnit 5: JUnit Platform, JUnit Jupiter, JUnit Vintage.
- Anotaciones: @Test, @DisplayName, @Disabled, @Nested, @Tag.

- Assertions: assertEquals, assertNotNull, assertThrows, assertAll, assertTimeout.
- Ciclo de vida: @BeforeEach, @AfterEach, @BeforeAll, @AfterAll.
- @ParameterizedTest con @ValueSource, @CsvSource, @MethodSource: pruebas con múltiples conjuntos de datos.
- Cobertura de código con JaCoCo; umbrales mínimos de cobertura en Maven.

Módulo 5 — Dobles de prueba con Mockito

- ¿Por qué mockear? Aislamiento de la unidad bajo prueba de sus dependencias.
- mock() y @Mock con @ExtendWith(MockitoExtension.class).
- Stubbing: when(mock.metodo()).thenReturn(valor), thenThrow(), thenAnswer().
- Verificación: verify(mock).metodo(argumento); times(n), atLeast(n), never().
- ArgumentCaptor: capturar argumentos pasados a los mocks para assertions.
- spy(): envoltura parcial de un objeto real; cuándo preferirlo sobre mock.

Módulo 6 — JDBC y patrón DAO

- JDBC: java.sql.DriverManager, Connection, Statement, PreparedStatement y ResultSet.
- Conexión a PostgreSQL (driver org.postgresql:postgresql) o MySQL (com.mysql.cj:mysql-connector-j).
- PreparedStatement: parámetros posicionales y prevención de SQL Injection.
- Manejo de transacciones: setAutoCommit(false), commit(), rollback().
- Patrón DAO (Data Access Object): separar la lógica de acceso a datos; interface + implementación JDBC.
- Connection pooling básico con HikariCP como dependencia Maven.

Módulo 7 — Maven

- Convención sobre configuración: estructura estándar de directorios (src/main/java, src/test/java).
- POM (Project Object Model): groupId, artifactId, version, packaging.
- Ciclo de vida: validate → compile → test → package → install → deploy.
- Gestión de dependencias: repositorio Maven Central, scope (compile, test, provided, runtime).
- Plugins esenciales: maven-compiler-plugin (versión Java), maven-surefire-plugin (JUnit 5), maven-shade-plugin (fat-JAR).
- Perfiles Maven: entornos de desarrollo, QA y producción con propiedades distintas.

Módulo 8 — JPMS, Reflexión, Anotaciones y proyecto final

- JPMS (Java 9+): module-info.java, requires, exports, opens, uses, provides.
- Reflexión: Class, Method, Field, Constructor; getDeclaredMethods(), invoke(), setAccessible().
- Anotaciones personalizadas: @interface, ElementType, RetentionPolicy.RUNTIME; procesamiento con reflexión.
- ServiceLoader: patrón Plugin para extensibilidad sin modificar el código base.
- Proyecto final integrador: framework de reportes extensible — módulo JPMS, plugins de formato (Strategy + ServiceLoader), pruebas unitarias con JUnit 5 + Mockito, persistencia con JDBC+DAO y empaquetado con Maven.

Evaluación

- Implementaciones de patrones por módulo (30%).
- Suite de pruebas automatizadas (30%).
- Proyecto final integrador con Maven + JDBC + JUnit 5 (40%).

Recursos y herramientas

- JDK 21 LTS, IntelliJ IDEA Community (soporte nativo para Maven y JUnit 5).
- Maven 3.9+, JUnit 5.10+, Mockito 5.x, HikariCP 5.x.
- PostgreSQL 16 o MySQL 8 como base de datos de práctica.
- Claude.ai para revisar implementaciones de patrones GoF, analizar cobertura de pruebas y sugerir refactorizaciones.