Java version	Features	Description
Java 8 LTS	Date/Time API.	Introduce un nuevo conjunto de clases en el paquete java.time para trabajar de manera más efectiva con fechas y tiempos, reemplazando a java.util.Date. Las clases como LocalDate, LocalTime, y LocalDateTime permiten una manipulación de fechas más intuitiva y menos propensa a errores.
	Streams.	Facilitan el procesamiento de datos de manera declarativa y funcional. Los Stream permiten operaciones como filtrar, mapear, y reducir grandes colecciones de datos de forma eficiente.
	Lambda Expressions.	Son funciones anónimas que permiten escribir sintaxis de código más conciso. Se usan principalmente para pasar comportamiento (funciones) como parámetros a métodos, facilitando el uso de la programación funcional.
	Method References.	Simplifican la escritura de lambdas cuando se quiere referenciar métodos o constructores existentes directamente mediante el uso de "::", facilitando la reutilización de código sin necesidad de crear una expresión lambda completa.
	Generalized Target-Type Inference.	Mejora la inferencia de tipos por parte del compilador. Permite que el compilador deduzca mejor el tipo genérico de métodos y expresiones lambda, reduciendo la cantidad de código explícito que se necesita escribir. La inferencia de tipos mejorada en Java 8 se aplica en múltiples contextos, como: Métodos genéricos, Constructores, Lambdas, Referencias a métodos, Asignaciones de variables locales y Colecciones. La clave es que el compilador de Java se ha vuelto más inteligente para deducir automáticamente los tipos, siempre que el contexto proporcione suficiente información para hacerlo.
	Functional Interfaces.	Son Interfaces que contienen un solo método abstracto, habilitando la programación funcional. Las expresiones lambda y referencias de método pueden ser usadas como instancias de estas interfaces. Ejemplos incluyen Runnable y Comparator.
	Optional Class.	Proporciona una forma de manejar valores que podrían ser nulos sin recurrir a verificaciones explícitas de nulos, mejorando la legibilidad y evitando errores. La clase Optional se utiliza para evitar el uso de null directamente, previniendo posibles excepciones NullPointerException.
	Comparable and Comparator Enhancements.	Se agregaron métodos estáticos y predeterminados en las interfaces Comparable y Comparator para simplificar la creación de comparadores. Métodos como comparing y thenComparing permiten la composición de comparadores de manera más sencilla.
	Add Collector.of() Method	Se introduce Collector.of() una herramienta versátil para construir colecciones personalizadas utilizando funciones específicas de combinación y finalización, el resultado de Collector.of() depende de cómo se definan estas funciones. No está orientado específicamente a la creación de colecciones inmutables y puede producir colecciones mutables o inmutables según la implementación del colector.
Java 11 LTS	Java Platform Module System.	El Java Platform Module System (JPMS) permite modularizar aplicaciones Java y la plataforma Java. Introducido en Java 9, permite agrupar clases y paquetes en módulos, lo que facilita la definición de dependencias entre módulos y especifica qué partes de un módulo son accesibles desde fuera. Esto mejora la encapsulación y ayuda a reducir el tamaño de las aplicaciones. En Java 11, se mejora la estabilidad del sistema de módulos.
	HTTP Client.	Java 11 estandariza el cliente HTTP introducido en Java 9, incorporándolo como una API estándar en el paquete java.net.http. Este cliente ofrece una forma más moderna y eficiente de realizar solicitudes y manejar respuestas, soportando tanto HTTP/1.1 como HTTP/2, y permitiendo conexiones WebSocket. En comparación con la antigua clase HttpURLConnection, la nueva API proporciona una mayor flexibilidad, facilita la programación asíncrona y mejora la eficiencia y la experiencia de desarrollo.
	Local Variable Type Inference.	Introducido en Java 10, permite declarar variables locales usando "var" para que el compilador infiera el tipo de variable automáticamente, simplificando la sintaxis. Por ejemplo, var list = new ArrayList<String>(); infiere que list es de tipo ArrayList<String>.
	Local-Variable Syntax for Lambda Parameters.	Permite usar "var" para declarar los parámetros de las expresiones lambda, mejorando la concisión del código sin perder tipado estático. Por ejemplo, puedes escribir "(var x, var y) -> x + y" en lugar de especificar el tipo explícitamente.
	Try-with-resources.	Introducido en Java 7, el bloque try-with-resources facilita el manejo automático de recursos (como streams o conexiones) que implementan AutoCloseable, asegurando su cierre al finalizar el bloque. En Java 11, esta característica se mejora para permitir que las variables que ya han sido declaradas fuera del bloque try-with-resources puedan ser utilizadas dentro del mismo, eliminando la necesidad de redeclararlas. Esto simplifica el código y mejora la legibilidad al evitar la duplicación de declaraciones.
	Java interface private method.	Los métodos privados en interfaces, introducidos en Java 9 y mantenidos en Java 11, permiten a los desarrolladores reutilizar código dentro de la interfaz sin exponer estos métodos a las clases que la implementan. Esta característica mejora la encapsulación al permitir que el código compartido sea utilizado por métodos predeterminados (default) dentro de la interfaz sin ser visible o accesible desde fuera de la interfaz.
	Factory Methods for Collections.	Nuevos métodos estáticos de fábrica, como List.of(), Set.of(), y Map.of(), para crear colecciones inmutables de manera sencilla, mejorando la legibilidad y seguridad del código.

		Por ejemplo, List.of(1, 2, 3) crea una lista inmutable con los elementos proporcionados, mientras que Set.of() y Map.of() hacen lo mismo para conjuntos y mapas, respectivamente. Esta mejora facilita la escritura de código más seguro y claro al evitar la necesidad de usar construcciones más complejas o manuales.
	Add copy factory methods for unmodifiable List, Set, Map.	Nuevos métodos de fábrica como List.copyOf(), Set.copyOf(), y Map.copyOf() que facilitan la creación de copias inmutables de colecciones existentes, asegurando que los datos originales no se modifiquen accidentalmente. Por ejemplo, List.copyOf(existingList) crea una copia inmutable de la lista proporcionada, preservando la integridad de los datos y evitando cambios no deseados en la colección original.
	New methods for unmodifiable collections in the Stream package (Collector Class).	Nuevos métodos en la clase Collectors del paquete java.util.stream, como Collectors.toUnmodifiableList(), Collectors.toUnmodifiableSet(), y Collectors.toUnmodifiableMap(). Estos métodos permiten crear colecciones inmutables directamente a partir de streams, simplificando su construcción y mejorando la seguridad del código al evitar modificaciones accidentales. Por ejemplo, Collectors.toUnmodifiableList() crea una lista inmutable a partir de un stream de elementos.
	New "Collection.toArray(IntFunction)" Default Method.	Un nuevo método en la interfaz Collection, toArray(IntFunction<T[]>), que facilita la conversión de colecciones en arrays de tipos específicos. Este método usa una función que proporciona tanto el tamaño como el tipo del array deseado, mejorando así la flexibilidad y eficiencia en el manejo de colecciones y arrays en Java.
	Time-Based Release Versioning.	Java adopta un esquema de versiones basado en tiempo, como Java 11 y Java 12, garantizando una nueva versión cada seis meses. Este enfoque facilita la planificación y adopción de nuevas características al nombrar las versiones según el año y el mes de su lanzamiento.
Java 17 LTS	Records.	Los records permiten definir clases inmutables orientadas a la transferencia de datos de manera eficiente. Al utilizar records, se automatizan tareas repetitivas, como la generación de los métodos equals(), hashCode(), y toString(), así como un constructor que inicializa los campos. Además, se crean automáticamente métodos getters para acceder a los valores de los campos, mientras que los setters no son necesarios debido a la inmutabilidad intrínseca de los records. Esta característica simplifica la creación de clases enfocadas en datos y mejora la claridad y mantenibilidad del código.
	Sealed Classes.	Las clases selladas permiten especificar qué otras clases pueden heredar de una clase o implementar una interfaz determinada. Esta característica ofrece un control más preciso sobre la jerarquía de clases, permitiendo restringir y prever el comportamiento del sistema. Con las clases selladas, se asegura que solo un conjunto definido de clases pueda extender o implementar la clase o interfaz sellada, promoviendo un diseño más seguro y predecible
	Switch Expressions.	En Java 17, las expresiones switch se estandarizan tras su introducción en Java 12, ofreciendo una sintaxis más moderna y concisa. A diferencia del switch tradicional, las expresiones switch pueden devolver un valor y permiten el uso de expresiones de múltiples líneas. La nueva sintaxis incluye el operador "->" para simplificar la definición de casos y mejorar la legibilidad del código. Esta actualización facilita la escritura de estructuras condicionales más expresivas y compactas.
	Pattern Matching for instanceof.	Introducido como característica preliminar en Java 14, estandarizado en Java 16, y continuado en Java 17. Este patrón simplifica el uso de "instanceof" al combinar la verificación de tipo y el casting en una sola operación. Si una expresión cumple con el tipo especificado, instanceof realiza automáticamente el casting y asigna el valor a una variable local, eliminando la necesidad de castings explícitos. Aunque en Java 17 no se añadieron mejoras significativas a esta funcionalidad, sigue siendo parte integral del lenguaje, mejorando la legibilidad y seguridad del código al evitar errores de conversión y reduciendo código redundante.
	Text Blocks.	Los bloques de texto, estandarizados en Java 17 tras haber sido introducidos como característica preliminar en versiones anteriores. Permiten representar cadenas de texto multilínea de forma más legible y conveniente. Estos bloques eliminan la necesidad de concatenar múltiples cadenas o incluir complicadas secuencias de escape, como las usadas para saltos de línea o comillas. Utilizando comillas triples (""""), es posible definir textos extensos con el formato deseado, lo que facilita la escritura de contenido como JSON, XML, o consultas SQL sin necesidad de ajustes adicionales.
	Add Stream.toList() method.	En Java 16 se añadió el método toList() a la interfaz Stream, lo cual simplifica la recolección de elementos de un flujo en una lista. Este método ofrece una forma conveniente y directa de convertir un stream en una lista inmutable, sin necesidad de usar un colector adicional como Collectors.toList(). A diferencia de Collectors.toList(), que puede devolver una lista mutable, Stream.toList() garantiza que la lista resultante sea inmutable, mejorando la seguridad en aplicaciones que requieren evitar modificaciones posteriores
	Add a new static method teeing to java.util.stream.Collectors.	El método teeing(), introducido en Java 12, permite combinar dos colectores (Collectors) en uno solo, lo que facilita recolectar datos de un Stream de dos formas diferentes al mismo tiempo. Funciona dividiendo los elementos de un flujo en dos coleccionadores, procesando cada conjunto de manera independiente, y luego combinando los resultados en un solo valor final. Este método es especialmente útil cuando necesitas obtener múltiples agregados a partir de un mismo conjunto de datos sin tener que procesar el flujo dos veces, mejorando tanto la eficiencia como la claridad del código.
	Adding support for the compact/short number formatting in JDK.	Se añade soporte en el JDK para el formato compacto de números, lo que permite representar valores numéricos grandes de manera más concisa y legible. En lugar de mostrar números completos, como "1200", se pueden abreviar a "1.2K" para miles, "1M" para millones, etc. Esta funcionalidad es

		especialmente útil en contextos donde el espacio es limitado, como interfaces de usuario o informes, proporcionando una representación clara y eficiente de los datos numéricos sin perder precisión significativa.
	Add a String instance method String::transform".	El nuevo método transform() en la clase String permite aplicar una función a una cadena de texto y obtener el resultado de dicha transformación. Este método simplifica la manipulación de cadenas al aceptar una función lambda o un método de referencia que realiza la transformación deseada. En lugar de escribir código extenso para modificar una cadena, transform() proporciona una forma más concisa y directa de realizar operaciones comunes, como conversión de mayúsculas/minúsculas, ajustes de formato, y otras transformaciones específicas. Esto mejora la legibilidad y flexibilidad del código.
	Add a new method "String::indent".	El método indent() ajusta la indentación de una cadena de texto añadiendo o eliminando espacios en blanco al inicio de cada línea. Al proporcionar un número específico de espacios, indent() permite aumentar o reducir la sangría del texto, lo que facilita la alineación y organización del contenido. Esta funcionalidad es especialmente útil para formatear bloques de texto multilínea, proporcionando una presentación más clara y ordenada, ideal para mejorar la legibilidad en documentos y código fuente.
	Enhanced Pseudo-Random Number Generators.	Se han añadido nuevas interfaces y clases para generadores de números pseudoaleatorios, lo que mejora significativamente tanto la flexibilidad como el rendimiento en la generación de números aleatorios. Estas actualizaciones proporcionan una gama más amplia de algoritmos, permitiendo a los desarrolladores seleccionar entre diversas estrategias para satisfacer necesidades específicas. La nueva API ofrece un control más detallado sobre la generación de números aleatorios, optimizando así el rendimiento y la versatilidad en aplicaciones que requieren alta calidad en la aleatorización.
Java 21 LTS	Simple Web Server.	Introduce una API que permite lanzar un servidor HTTP básico, diseñado para facilitar el desarrollo y la prueba de aplicaciones web. Este servidor es ideal para la creación de prototipos y pruebas rápidas, ya que evita la necesidad de configurar un servidor completo. Es una herramienta práctica para desarrolladores que buscan una solución sencilla y eficiente para validar sus aplicaciones en un entorno controlado. No obstante, debido a su diseño simplificado, el Simple Web Server presenta algunas limitaciones. No proporciona soporte completo para protocolos avanzados como HTTP/2 ni para conexiones seguras HTTPS. Aunque permite manejar solicitudes HTTP y puede ser útil para implementar servicios REST y SOA en entornos de desarrollo, no cuenta con las funcionalidades avanzadas de servidores web más robustos como Apache Tomcat o Jetty. Para necesidades específicas de protocolo y seguridad, se recomienda usar servidores web que ofrezcan soporte completo para estas características.
	Sequenced Collections.	Java 21 introduce interfaces para colecciones secuenciales: SequencedCollection, SequencedMap, y SequencedSet. Estas interfaces mejoran el manejo de datos ordenados con métodos adicionales. SequencedCollection: Añade first(), last(), subList(), y forEachIndexed() para manipular y acceder a elementos secuenciales. SequencedMap: Incluye firstEntry(), lastEntry(), y descendingMap() para trabajar con entradas en mapas secuenciales. SequencedSet: Ofrece first(), last(), y subSet() para gestionar conjuntos secuenciales. Las clases concretas como ArrayList, LinkedList, TreeMap, TreeSet, y ArrayDeque implementan estas interfaces, mejorando la eficiencia en la inserción, eliminación y acceso a elementos en posiciones específicas. Estas mejoras hacen que trabajar con colecciones secuenciales sea más eficiente y expresivo.
	Pattern Matching for switch.	Java 21 amplía el patrón de coincidencia en las declaraciones switch, ofreciendo un uso más flexible y conciso. Esta mejora facilita la escritura de casos complejos al permitir patrones más expresivos y simplificar la extracción de datos de objetos. Ahora puedes inferir el tipo de objeto y extraer sus componentes directamente en la expresión switch, mejorando la legibilidad y mantenibilidad del código.
	Record Patterns.	Java 21 mejora los registros (records) al introducir patrones de coincidencia avanzados. Ahora, estos patrones pueden usarse en declaraciones switch y otros contextos, como expresiones instanceof y asignaciones. Esto simplifica el desempaquetado y la extracción de valores de los registros, permitiendo un manejo de datos más claro y conciso.
	Virtual Threads.	Java 21 introduce los hilos virtuales, gestionados por la máquina virtual de Java (JVM), que ofrecen una solución más eficiente para la programación concurrente en comparación con los hilos tradicionales, los cuales son gestionados por el sistema operativo. Gracias a su gestión por parte de la JVM, los hilos virtuales reducen la sobrecarga de recursos, mejorando tanto la eficiencia como la escalabilidad en comparación con los hilos tradicionales. En Java, los hilos tradicionales se crean con java.lang.Thread o java.lang.Runnable y se gestionan a través de java.util.concurrent.Executors utilizando métodos como start(), run(), y submit(). Los hilos virtuales, introducidos en Java 21, se crean con Thread.ofVirtual() y se gestionan mediante java.util.concurrent.Executors.newVirtualThreadPerTaskExecutor() o Executors.newThreadPerTaskExecutor(). La clase ForkJoinPool gestiona tareas concurrentes de manera más eficiente al usar hilos virtuales, mientras que CompletableFuture facilita la programación asincrónica ejecutando tareas en hilos virtuales a través de un ExecutorService.
	Internet-Address Resolution SPI.	Java 21 introduce un Service Provider Interface (SPI) para la resolución de direcciones de Internet, que permite personalizar el proceso de resolución de nombres y direcciones de red. Esto se logra definiendo una interfaz AddressResolver y una clase que la implemente, como DefaultAddressResolver. La implementación se registra mediante un archivo en META-INF/services con el nombre de la interfaz y la clase implementadora. Luego, el ServiceLoader carga y utiliza esta implementación, proporcionando flexibilidad y adaptabilidad en la resolución de direcciones de red.

	Key Encapsulation Mechanism API.	En Java 21, la API de encapsulación de claves en el módulo java.security ha sido mejorada para optimizar la criptografía y la gestión de claves en aplicaciones seguras. Se han introducido nuevas interfaces, como KeyManagerFactorySpi, para permitir proveedores personalizados de claves, mientras que las clases existentes como Key, KeyStore, SecretKey, Cipher, y KeyPairGenerator han sido actualizadas para ofrecer un rendimiento más eficiente y una mayor seguridad. Estas mejoras facilitan la generación, almacenamiento, cifrado y descifrado de claves y datos, garantizando un manejo más seguro y controlado de la información sensible.
	Prepare to Disallow the Dynamic Loading of Agents.	Java 21 establece las bases para deshabilitar la carga dinámica de agentes en futuras versiones. Los agentes son componentes que pueden ser cargados y ejecutados en tiempo de ejecución para tareas como monitoreo o modificación del comportamiento de la aplicación. Al prepararse para desactivar esta carga dinámica, Java 21 mejora la seguridad al reducir riesgos asociados con la introducción de código externo no verificado y aumenta la estabilidad al asegurar un entorno de ejecución más predecible y menos susceptible a cambios inesperados. Esta medida busca proteger el entorno de ejecución y garantizar una mayor seguridad y confiabilidad para las aplicaciones.
Java 22	Foreign Function & Memory API.	La Foreign Function & Memory API de Java 22, ubicada en el paquete java.foreign, permite a los programas Java interactuar de manera segura y eficiente con código y datos fuera de la JVM, eliminando la necesidad de JNI. Ofrece dos componentes principales: la Foreign Function API, que facilita la invocación de funciones nativas a través de interfaces que definen la firma de las funciones, y la Foreign Memory API, que permite acceder y manipular memoria fuera del heap de la JVM mediante clases como MemorySegment y MemoryAccess. Esta API simplifica la interoperabilidad con bibliotecas nativas y mejora el rendimiento al reducir el overhead asociado con JNI y gestionar de manera más segura la memoria externa.
	Unnamed Variables & Patterns.	La característica de Unnamed Variables & Patterns en Java 22 permite usar el guion bajo (_) para representar variables y patrones sin nombres explícitos, lo que mejora la concisión y legibilidad del código. Esta funcionalidad, integrada en la sintaxis de patrones, se aplica en bloques switch y métodos para manipular datos, simplificando la codificación de patrones complejos al evitar nombres temporales innecesarios. Por ejemplo, en un bucle for como "var (_, age) = person;", se usa "_" para ignorar el primer valor de la tupla y extraer solo el valor de "age".
	Launch Multi-File Source-Code Programs.	La característica Launch Multi-File Source-Code Programs en Java 22 permite ejecutar aplicaciones distribuidas en varios archivos fuente directamente desde la línea de comandos o herramientas de desarrollo, sin compilar previamente cada archivo por separado. Usando el comando java junto con opciones como --sourcepath para especificar el directorio de los archivos fuente y -cp para definir el classpath, esta funcionalidad, implementada en el paquete java.compiler, simplifica el desarrollo y las pruebas de aplicaciones complejas al evitar compilaciones manuales extensas.
Java 23	Markdown Documentation Comments	La nueva característica de Markdown Documentation Comments en Java 23 permite el uso de sintaxis Markdown en los comentarios de documentación Javadoc, lo que mejora significativamente la legibilidad y funcionalidad de la documentación generada. Esta mejora se integra directamente en la herramienta javadoc del JDK, sin requerir modificaciones en paquetes o clases específicas, y se centra en cómo se escriben y procesan los comentarios Javadoc.
	ZGC: Generational Mode by Default	En Java 23, el recolector de basura ZGC (Garbage Collector Z) ahora utiliza el modo generacional por defecto, en lugar del modo no generacional que empleaba anteriormente. Este cambio organiza los objetos en dos generaciones, joven y vieja, lo que optimiza el proceso de recolección de basura. La configuración se realiza mediante la opción de línea de comandos -XX:+UseZGC y parámetros asociados con la recolección generacional. La implementación y gestión de estos cambios se encuentran en el paquete jdk.internal.vm, con detalles específicos manejados internamente por el recolector ZGC.
	Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal	En Java 23, los métodos de acceso a memoria en la clase sun.misc.Unsafe han sido marcados como obsoletos (deprecated) y se planea su eliminación en futuras versiones. Estos métodos, que permiten operaciones de bajo nivel para acceder a memoria directa y manipular datos (como putInt(Object o, long offset, int x) y getLong(Object o, long offset)), están ubicados en el paquete sun.misc. La eliminación de estos métodos se detalla en la documentación de sun.misc.Unsafe, y se recomienda usar APIs estándar para operaciones de memoria.