Parfait. Je vais compiler une veille Java complète couvrant les nouveautés, annonces et évolutions majeures du mois de mars 2025. Cela inclura les mises à jour du JDK, les évolutions des frameworks populaires, les nouvelles bibliothèques, les événements communautaires et les meilleures pratiques mises en avant. Je te reviens très bientôt avec une synthèse structurée et des exemples concrets.

Veille Java – Mars 2025

Synthèse du mois de mars 2025

Le mois de mars 2025 a été particulièrement riche en actualités pour l'écosystème Java. La sortie officielle de **Java 24** a marqué un jalon important, avec de nombreuses évolutions du JDK tant au niveau du langage que de la plateforme. Les principaux **frameworks Java** ont également connu des mises à jour notables : Spring prépare sa prochaine génération tout en publiant des correctifs et versions milestones, Jakarta EE 11 approche de sa finalisation, et des frameworks cloud-native comme Quarkus et Micronaut ont livré de nouvelles versions stables. Du côté des **outils et bibliothèques**, on observe l'émergence de solutions pour améliorer la productivité des développeurs (API plus fluides, outils d'observabilité, intégration de l'IA, etc.). La **communauté Java** a vibré au rythme de conférences (JavaOne, etc.), d'annonces officielles (Oracle) et de partages de meilleures pratiques (par exemple autour des threads virtuels, de l'observabilité et de l'architecture modulaire). Nous proposons dans cette veille une synthèse détaillée : mises à jour du JDK par version, évolutions des frameworks majeurs (Spring, Jakarta EE, Micronaut, Quarkus), nouveautés des bibliothèques/outils, faits marquants de la communauté, conseils et retours d'expérience, exemples de code illustratifs, ainsi qu'une analyse critique de l'impact de ces nouveautés et un point sur les roadmaps officielles.

Mises à jour du JDK (Java 17, 21, 22, 24…)

Java 24 : une nouvelle version majeure du JDK

Java 24 (JDK 24) a atteint sa disponibilité générale le 18 mars 2025 (JDK 24 (https://openjdk.org/projects/jdk/24/#:%7E:text=JDK%2024%20reached%20General%20Availability,other%20vendors%20will%20follow%20shortly)). Cette version s'avère particulièrement riche, totalisant **24 JEPs (propositions d'amélioration)** – un record depuis l'adoption du calendrier de sortie semestriel (Six JDK 24 Features You Should Know About - Azul | Better Java Performance, Superior Java Support (https://www.azul.com/blog/six-jdk-24-features-you-should-know-about/#:%7E:text=%28JEPs%29)). Parmi ces JEPs, environ la moitié introduisent des fonctionnalités **définitives** (finalisées après plusieurs itérations en incubateur ou preview), tandis que l'autre moitié propose des fonctionnalités encore **en incubation ou en mode preview** (expérimentales) (Java 24 Delivers New Experimental and Many Final Features - InfoQ (https://www.infoq.com/news/2025/03/java24-released/#:%7E:text=match%20at%20L293%20Two%20of,the%20incubation%20and%20preview%20processes)). Java 24 apporte des améliorations dans de nombreux domaines du langage et de la JVM : performances, concurrence, cryptographie, productivité du code, etc. Voici quelques faits marquants :

- **Performance de la JVM et du GC** : introduction du **GC Shenandoah générationnel** (JEP 404) en mode expérimental, pour améliorer le ramassage de déchets en divisant le tas en générations (Java 24 Delivers New Experimental and Many Final Features - InfoQ (https://www.infoq.com/news/2025/03/java24-released/#:%7E:text=,Late%20Barrier%20Expansion%20for%20G1)). De même, le JDK intègre les **entêtes d'objets compactés** (JEP 450, expérimental) réduisant l'empreinte mémoire des objets Java (Java 24 Delivers New Experimental and Many Final Features - InfoQ (https://www.infoq.com/news/2025/03/java24-released/#:%7E:text=,Late%20Barrier%20Expansion%20for%20G1)). Ces évolutions de bas niveau visent à optimiser les performances et la consommation mémoire des applications sans nécessiter de modification de code applicatif. Par ailleurs, le ramasse-miettes **ZGC supprime son mode non-générationnel** pour se focaliser sur un mode générationnel unique plus efficace (JDK 24 (https://openjdk.org/projects/jdk/24/#:%7E:text=,Synchronize%20Virtual%20Threads%20without%20Pinning)).

- **Langage Java et productivité** : Java 24 continue d'enrichir le langage. En particulier, le **pattern matching** s'étend désormais aux types primitifs (JEP 488, en deuxième preview) (Java 24 Delivers New Experimental and Many Final Features - InfoQ (https://www.infoq.com/news/2025/03/java24-released/#:%7E:text=,Second%20Preview)). Concrètement, il est maintenant possible d'utiliser `instanceof` et `switch` directement avec des types primitifs, là où auparavant seuls les types objets étaient admis. Par exemple, le code ci-dessous montre l'utilisation de `instanceof` avec un entier :

  ```
  Object value = 42;
  if (value instanceof int i) {         // le JDK 24 permet un pattern matching sur int
      System.out.println("Valeur entière : " + i);
  }
  ```

  Et de même, un `switch` peut matcher un type primitif avec un cas de garde générique :

  ```
  int status = 2;
  String message = switch (status) {
      case 0 -> "OK";
      case 1 -> "Warning";
      case 2 -> "Error";
      case int i -> "Status inconnu : " + i;
  };
  System.out.println(message);  // "Status inconnu : 2"
  ```

  Ces évolutions éliminent des conversions ou wrappers inutiles et rendent le code plus expressif (JDK 24 New Features Every Java Developer Must Know - DEV Community (https://dev.to/myexamcloud/jdk-24-new-features-every-java-developer-must-know-21dj#:%7E:text=JDK%2024%20extends%20pattern%20matching,expressive%20and%20eliminating%20unnecessary%20casts)) (JDK 24 New Features Every Java

Developer Must Know - DEV Community (https://dev.to/myexamcloud/jdk-24-new-features-every-java-developer-must-know-21dj#:%7E:text=Example%3A%20,Patterns)). Java 24 introduit également la **syntaxe de "corps de constructeurs flexibles"** (JEP 492, troisième preview) permettant d'exécuter du code avant l'appel à un constructeur parent via `super()` ou `this()`. Cela offre plus de naturel pour valider des arguments ou initialiser des champs avant de chaîner les constructeurs, tout en maintenant les garanties d'initialisation de Java (Oracle Releases Java 24 (https://www.oracle.com/news/announcement/oracle-releases-java-24-2025-03-18/#:%7E:text=from%20support%20of%20primitive%20types,reliable%20when%20methods%20are%20overridden)). Autre amélioration notable pour la lisibilité : la **déclaration d'imports de module** (JEP 494, deuxième preview) qui autorise d'importer en une seule directive toutes les classes publiques d'un module, simplifiant l'utilisation des bibliothèques modulaires (Oracle Releases Java 24 (https://www.oracle.com/news/announcement/oracle-releases-java-24-2025-03-18/#:%7E:text=a%20class%20more%20reliable%20when,inference%2C%20library%2C%20or%20service%20calls)). Enfin, on note la quatrième preview de la fonctionnalité *"source files simples et méthodes `main` d'instance"* (JEP 495) qui vise à simplifier l'écriture de petits programmes ou scripts Java, notamment pour les débutants, en autorisant des classes sans déclaration de package et en permettant une méthode `main` non statique (Oracle Releases Java 24 (https://www.oracle.com/news/announcement/oracle-releases-java-24-2025-03-18/#:%7E:text=fundamental%20Java%20classes%20without%20needing,expand%20their%20programs%20with%20more)).

- **Concurrence et parallélisme** : les apports du **Projet Loom** continuent de se faire sentir. Java 21 avait généralisé les **threads virtuels** (aussi appelés *Virtual Threads*), et Java 24 améliore encore leur intégration. Le JEP 491 **"Synchronize Virtual Threads without Pinning"** élimine la limitation qui faisait qu'un bloc `synchronized` pouvait "épingler" un thread virtuel sur un thread OS, ce qui annulait son avantage. Désormais, la synchronisation intrinsèque fonctionne sans bloquer la mise en pause du thread virtuel, ce qui permet d'utiliser les `synchronized` de manière transparente avec les threads virtuels (Six JDK 24 Features You Should Know About - Azul | Better Java Performance, Superior Java Support (https://www.azul.com/blog/six-jdk-24-features-you-should-know-about/#:%7E:text=JEP%20491%3A%20Synchronize%20virtual%20threads,without%20pinning)). Par ailleurs, Java 24 inclut la **quatrième preview de la Concurrence Structurée** (JEP 499) (Java 24 Delivers New Experimental and Many Final Features - InfoQ (https://www.infoq.com/news/2025/03/java24-released/#:%7E:text=match%20at%20L290%20,bit%20x86%20Port%20for%20Removal)). La Concurrence Structurée introduit une classe de scope `StructuredTaskScope` pour gérer un groupe de tâches concurrentes comme une unité unique, facilitant la gestion des erreurs et l'annulation groupée de threads enfants. Elle *"permet de traiter un ensemble de tâches liées s'exécutant en parallèle comme une unité de travail, simplifiant ainsi la gestion des erreurs, l'annulation et améliorant la fiabilité et l'observabilité du code multithread"* (Java 24 Delivers New Experimental and Many Final Features - InfoQ (https://www.infoq.com/news/2025/03/java24-released/#:%7E:text=feature%20simplifies%20concurrent%20programming%20by,of%20the%20proposed%20API%20changes)). Bien qu'encore en mode preview, cette approche tend à devenir un **pattern de concurrence recommandé**, en particulier combinée aux threads virtuels pour obtenir du parallélisme efficace avec un code au style séquentiel.

- **APIs de bibliothèque et sécurité** : Plusieurs nouvelles APIs font leur apparition. Le JDK propose une API standard pour manipuler les fichiers de classes Java (**Class-File API**, JEP 484) afin de faciliter la génération ou analyse dynamique de bytecode (Oracle Releases Java 24 (https://www.oracle.com/news/announcement/oracle-releases-java-24-2025-03-18/#:%7E:text=efficient%20in%20reading%2C%20writing%2C%20and,Fourth%20Preview%29%3A%20Helps%20developers)). Côté flux de données, les **Stream Gatherers** (JEP 485) introduisent de nouvelles opérations intermédiaires pour les streams, permettant des transformations de données plus flexibles que les opérations fournies jusqu'alors (Oracle Releases Java 24 (https://www.oracle.com/news/announcement/oracle-releases-java-24-2025-03-18/#:%7E:text=,API%20for%20parsing%2C%20generating%2C%20and)). En cryptographie, Java 24 se positionne face aux menaces futures en intégrant des algorithmes résistants aux ordinateurs quantiques : le JDK ajoute une API de **fonction de dérivation de clés (KDF)** en mode preview (JEP 478) ainsi que l'implémentation de mécanismes de chiffrement et de signature basés sur des systèmes à treillis (JEP 496 et JEP 497) (JDK 24 (https://openjdk.org/projects/jdk/24/#:%7E:text=,Second%20Preview)) (JDK 24 (https://openjdk.org/projects/jdk/24/#:%7E:text=%3E%20493%3A%20Linking%20Run,501%3A%20%2029)). Ces outils « post-quantiques » permettent dès maintenant de tester des solutions de chiffrement censées résister aux futures capacités de calcul quantique. Enfin, notons que Java 24 marque la **désactivation définitive du Security Manager** (JEP 486) (Java 24 Delivers New Experimental and Many Final Features - InfoQ (https://www.infoq.com/news/2025/03/java24-released/#:%7E:text=,Second%20Preview)). Cet ancien mécanisme de sandboxing, déjà déprécié depuis Java 17, est désormais entièrement inopérant pour des raisons de sécurité et d'entretien du code (son usage ayant fortement décliné avec la fin des Applets et l'évolution des pratiques de déploiement) (Java News Roundup: Jakarta EE 11 Web Profile, GlassFish, TornadoVM, Micronaut, JHipster, Applet API - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-mar31-2025/#:%7E:text=,effort%20to%20refactor%20the%20TCK)). À ce sujet, Oracle a également lancé un JEP candidat pour **supprimer purement et simplement l'API Applet** du JDK, celle-ci étant obsolète depuis des années (Java News Roundup: Jakarta EE 11 Web Profile, GlassFish, TornadoVM, Micronaut, JHipster, Applet API - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-mar31-2025/#:%7E:text=This%20week%27s%20Java%20roundup%20for,to%20remove%20the%20Applet%20API)).

En synthèse, Java 24 se présente comme une version transitoire riche en fonctionnalités : elle finalise plusieurs améliorations amorcées dans les versions 21 à 23 (GC optimisés, threads virtuels mieux intégrés, patterns étendus…), et introduit en avant-première de nouvelles capacités (concurrence structurée, cryptographie post-quantique, etc.) qui deviendront peut-être standard dans les prochaines versions. Il convient de noter que Java 24 **n'est pas une version LTS** (support long terme) – la prochaine LTS sera Java 25, attendue pour septembre 2025. Ainsi, de nombreuses entreprises resteront possiblement sur Java 17 ou Java 21 en production, tout en évaluant ces nouveautés de Java 24 pour préparer l'avenir.

État des versions LTS (Java 17 et Java 21) et précédentes

- **Java 21 (LTS)** : Sortie en septembre 2023, Java 21 est la dernière version LTS en date et gagne progressivement en adoption en ce début 2025. Elle a introduit notamment les *Virtual Threads* de Loom en version finale, les *Record Patterns*, les *Scoped Values*, *Structured Concurrency* (en preview initiale) et bien d'autres améliorations. Au mois de mars 2025, Java 21 bénéficie déjà de correctifs de maintenance : la version **21.0.2** (CPU de janvier 2024) est disponible, et le prochain correctif 21.0.3 est planifié pour avril 2025 dans le cadre du cycle trimestriel des patchs de sécurité d'Oracle. Aucune nouvelle fonctionnalité majeure n'est ajoutée sur la branche 21, mais des **mises à jour de sécurité et de stabilité** ont été livrées. Les développeurs sont encouragés à migrer vers Java 21 LTS s'ils utilisent encore Java 17, afin de profiter de ces nouveautés stables sur une version supportée long terme. D'ailleurs, les frameworks majeurs (Spring 6/Spring Boot 3, Quarkus 3, Micronaut 4, Jakarta EE 11, etc.) ont aligné leur compatibilité sur Java 21, ce qui facilite la transition.

- **Java 17 (LTS)** : Toujours largement utilisé en production (LTS de 2021), Java 17 continue de recevoir des **mises à jour de maintenance**. La dernière version de correctif est **17.0.14** (janvier 2025) et apporte son lot de corrections de bugs et failles de sécurité. Java 17 reste la référence pour de nombreux projets en 2025, mais commence à accuser son retard fonctionnel (pas de virtual threads ni des nouveautés plus récentes). Le support de Java 17 par Oracle et OpenJDK se poursuit (prévu jusqu'en 2029 pour

Oracle Java 17), toutefois la communauté pousse à planifier la migration vers Java 21 LTS dans les prochains trimestres, pour éviter de manquer les avancées significatives apparues depuis Java 17.

- **Versions 18, 19, 20, 22, 23 (non-LTS)** : Ces versions intermédiaires, sorties tous les six mois, ont servi d'itérations pour introduire progressivement les fonctionnalités qui, pour la plupart, ont été consolidées dans Java 21 et Java 24. Par exemple, Java 19 et 20 avaient introduit en preview les virtual threads et le pattern matching étendu, Java 22 (mars 2024) a proposé en avant-première la fonctionnalité de *"String Templates"* (littéraux de modèle de chaîne de caractères pour faciliter la construction de chaînes structurées) et Java 23 (sept 2024) a continué l'incubation de la Concurrence Structurée et d'autres API. À ce jour, ces versions ne sont plus mises à jour activement (cycle de support court). Les utilisateurs ayant expérimenté sur Java 22 ou 23 sont incités à passer sur Java 24 ou à revenir sur une LTS, car les correctifs de sécurité ne sont pas garantis au-delà de quelques mois sur les non-LTS.

En résumé, le JDK en mars 2025 voit une cohabitation entre Java 17 et Java 21 (LTS en production) et Java 24 (dernier cru non-LTS pour les plus aventureux). Java 24 offre un aperçu du futur de la plateforme, tandis que la communauté prépare déjà le terrain pour **Java 25 LTS** prévu en fin d'année 2025, qui devrait intégrer la maturité des fonctionnalités aujourd'hui en preview. Nous verrons en conclusion l'impact de ces évolutions sur la feuille de route des projets Java.

Évolutions des frameworks Java majeurs

Les principaux frameworks et plateformes Java ont connu en mars 2025 des évolutions significatives, reflétant l'adoption progressive des nouveautés du JDK et les besoins changeants (cloud, microservices, modularité, etc.). Voici un tour d'horizon pour **Spring**, **Jakarta EE**, **Quarkus** et **Micronaut**.

Spring Framework & Spring Boot

L'écosystème Spring continue d'évoluer à un rythme soutenu. En mars 2025, pas de révolution majeure du côté des versions stables, mais **plusieurs releases correctives et des versions milestones** ont été annoncées, ainsi que des indices sur la prochaine génération du framework.

- **Spring Boot 3.4.x et 3.5.0 (milestone)** : La famille Spring Boot 3 (basée sur Spring Framework 6) s'affine. Deux versions d'entretien sont sorties : **Spring Boot 3.4.4** et **3.3.10**, apportant essentiellement des corrections de bugs, des mises à jour mineures de dépendances et de la documentation (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=Similarly%2C%20the%20release%20of%20Spring,10)). Fait notable, Spring Boot a désactivé par défaut le support de l'**APR (Apache Portable Runtime) de Tomcat** lorsque l'application tourne sur un JDK 24 ou supérieur (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=Similarly%2C%20the%20release%20of%20Spring,10)). Cette modification évite des warnings du JDK liés à l'utilisation de JNI, Tomcat APR s'appuyant sur des bibliothèques natives plus vraiment nécessaires depuis les améliorations de performances de Java récentes. Parallèlement, le développement de Spring Boot 3.5 progresse : la **3e milestone de Spring Boot 3.5.0 (3.5.0-M3)** est sortie, embarquant un lot de nouveautés et d'améliorations (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20third%20milestone%20release%20of,found%20in%20the%20release%20notes)). Parmi celles-ci : un support amélioré d'OpenTelemetry (utilisation correcte de l'attribut `service.namespace` dans les traces), des améliorations pour Spring Batch (nouvelles propriétés de configuration) et l'ajout d'une classe utilitaire pour faciliter la configuration de services d'annuaire LDAP en mémoire dans les tests (`LdapDockerComposeConnectionDetailsFactory`) (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=new%20features%20such%20as%3A%20a,found%20in%20the%20release%20notes)). Toutes ces évolutions témoignent d'un engagement à améliorer l'expérience développeur (observabilité, tests, etc.) sur la plateforme Spring Boot actuelle. Spring Boot reste compatible Java 17 minimum et supporte pleinement Java 21 ; il est testé également sur Java 24 (d'où l'adaptation pour Tomcat+JDK24).

- **Spring Framework 6.2.x** : Du côté du *framework* central, la branche 6.2 a eu une mise à jour **Spring Framework 6.2.5** (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=)). Cette version apporte essentiellement des corrections et quelques petites améliorations de commodité : par exemple, la méthode `comment()` de l'API Server-Sent Events accepte désormais un commentaire vide (utile pour garder la connexion ouverte sans envoyer de données) (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20release%20of%20Spring%20Framework,found%20in%20the%20release%20notes)), et le convertisseur de formulaires `FormHttpMessageConverter` lance désormais une exception plus spécifique (`HttpMessageNotReadableException`) lorsque les données de formulaire sont invalides, ce qui aide les développeurs à mieux gérer les erreurs de requêtes malformées (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20release%20of%20Spring%20Framework,found%20in%20the%20release%20notes)). Ce sont des changements mineurs mais issus des retours terrain, améliorant la robustesse de l'ensemble.

- **Spring Cloud 2024.0 "Mooregate"** : L'écosystème Spring Cloud, regroupant les projets destinés aux architectures microservices, a publié sa **version 2024.0.1 (Mooregate)** en mars (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20release%20of%20Spring%20Cloud,found%20in%20the%20release%20notes)). Cette release Spring Cloud 2024.0 est basée sur Spring Boot 3.4.3 et intègre des corrections de bugs ainsi que des mises à jour de ses sous-projets : Spring Cloud Kubernetes 3.2.1, Spring Cloud Function 4.2.2, Spring Cloud OpenFeign 4.2.1, Spring Cloud Stream 4.2.1, Spring Cloud Gateway 4.2.1, etc. (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20release%20of%20Spring%20Cloud,found%20in%20the%20release%20notes)). Il s'agit de versions mineures alignées avec la plateforme Spring Boot 3.4. A noter que la prochaine ligne Spring Cloud sera **2025.0** (nom de code à thème londonien, probablement *"Northfields"* d'après les conventions précédentes) et sortira avec Spring Boot 3.5 en milieu d'année.

- **Spring Security 6.5.0 (milestone)** : Spring Security prépare sa prochaine évolution majeure (6.5). La **3e version milestone de Spring Security 6.5.0** a été livrée, apportant le support du RFC 9068 (Profile JWT pour les tokens OAuth2 Access Token) (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-

2025/#:%7E:text=The%20third%20milestone%20release%20of,found%20in%20the%20release%20notes)). Cela signifie une meilleure prise en charge native des **JSON Web Tokens** dans Spring Security, conformément aux derniers standards OAuth2 (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20third%20milestone%20release%20of,found%20in%20the%20release%20notes)). De plus, cette version déprécie l'interface `ConfigAttribute` (utilisée historiquement pour représenter les configurations de sécurité) pour encourager l'usage de nouvelles API sans ce type central, ce qui clarifie l'extension de Spring Security (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=for%20RFC%209068%2C%20JSON%20Web,found%20in%20the%20release%20notes)). Enfin, Spring Security 6.5 M3 introduit le support de la **propagation automatique du contexte** avec Micrometer, ce qui facilite le couplage entre la sécurité et l'observabilité (par exemple, associer des traces de monitoring à l'identité ou au contexte de sécurité courant) (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=for%20RFC%209068%2C%20JSON%20Web,found%20in%20the%20release%20notes)).

- **Autres projets Spring** : L'écosystème complet Spring étant très vaste, de nombreux autres modules ont eu des publications en mars. Citons par exemple :
  - **Spring Authorization Server 1.5.0-M2** : évolution du serveur d'autorisation OAuth2 de Spring, avec de petites améliorations (optimisation du service JDBC d'authentification, support des "Pushed Authorization Requests" RFC 9126) (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20second%20milestone%20release%20of,found%20in%20the%20release%20notes)).
  - **Spring for GraphQL 1.4.0-M1** : le projet d'intégration GraphQL propose un alignement avec la spécification GraphQL-over-HTTP et un meilleur support du fédération Apollo 5, signe de la maturité croissante de GraphQL côté Java (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=)).
  - **Spring Integration 6.5.0-M3** : apporte des filtres pour écarter les fichiers trop anciens et supprime des API obsolètes dans la gestion d'intégration de messages (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20third%20milestone%20release%20of,found%20in%20the%20release%20notes)) (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=dependency%20upgrades%20and%20new%20features,found%20in%20the%20release%20notes)).
  - **Spring Modulith 1.4.0-M3** : ce projet récent visant à encourager l'architecture modulaire dans les applications monolithiques Java continue d'évoluer. La milestone 1.4.0 améliore le support des tests modulaires (`@ApplicationModuleTest`) et l'assertion d'événements publiés pour renforcer les contrats modulaires (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20third%20milestone%20release%20of,found%20in%20the%20release%20notes)). Des correctifs de performance ont également été apportés dans les versions 1.3.4 et 1.2.10 de Modulith (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=Similarly%2C%20the%20release%20of%20Spring,10)). Le fait que Spring investisse sur Modulith montre la popularité grandissante des **monolithes modulaires** comme alternative aux microservices dans certains cas.
  - **Spring Batch 5.2.2** : mise à jour du framework de traitements par lots, qui ajoute notamment des hints AOT (Ahead-of-Time) pour faciliter la compilation native et marque certaines interfaces avec `@FunctionalInterface` pour améliorer la clarté du code (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20release%20of%20Spring%20Batch,found%20in%20the%20release%20notes)).
  - **Spring AMQP 4.0.0-M2** : prépare la prochaine version majeure du support RabbitMQ, en ajoutant le support du protocole **AMQP 1.0** (via un nouveau module client) et du RPC simplifié via une nouvelle classe `RabbitAmqpTemplate` (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20second%20milestone%20release%20of,found%20in%20the%20release%20notes)).
  - **Spring for Apache Kafka 4.0.0-M1** : milestone alignée sur Spring Framework 7 (donc encore au stade expérimental), qui migre toutes les annotations de nullabilité vers **JSpecify** (effort de standardisation de la null-safety) et améliore les performances de certaines opérations de consommation Kafka (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=The%20first%20milestone%20release%20of,on%20this%20release%20may%20be)).
  - **Spring for Apache Pulsar 1.2.4** : support de Pulsar (système de messagerie streaming) mis à jour dans Spring, avec montée de version vers Jakarta EE 10 (Spring 6.2.4) et Micrometer 1.14, inclus dans Spring Boot 3.4.4 (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=)).
  - **Spring Web Services 4.1.0-M1** : ajout du support d'Axiom 2.0 (lib XML) compatible Jakarta EE, et déprécation de certaines anciennes classes de configuration au profit de méthodes par défaut plus simples (Spring News Roundup: Milestone Releases of Boot, Security, Auth Server, GraphQL, Integration, AMQP - InfoQ (https://www.infoq.com/news/2025/03/spring-news-roundup-mar17-2025/#:%7E:text=)).

En somme, **Spring en mars 2025 consolide sa plateforme 6.x/Boot 3.x**, avec un accent sur les correctifs et la qualité, tout en préparant la suite via des versions milestone. La compatibilité avec les environnements modernes est sans cesse améliorée (observabilité OpenTelemetry, adaptation au JDK 24, support de Jakarta EE 10/11 dans les intégrations). Un point marquant est l'annonce de la prochaine génération : **Spring Framework 7.0** est officiellement planifié pour **fin 2025** (novembre) (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=At%20the%20same%20time%2C%20we,0)) (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=We%20will%20upgrade%20our%20baseline,the%20recently%20released%20JSpecify%20annotations)). D'après les informations partagées par l'équipe Spring, Spring 7.0 élèvera la baseline à Jakarta EE 11 (Tomcat 11, Hibernate ORM 7, etc.) et « embrassera » le JDK 25 LTS, tout en maintenant une compatibilité minimale avec le JDK 17 pour accompagner la transition de l'écosystème (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=We%20will%20upgrade%20our%20baseline,the%20recently%20released%20JSpecify%20annotations)) (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=We%20will%20upgrade%20our%20baseline,the%20recently%20released%20JSpecify%20annotations)). Spring 7 devrait aussi intégrer nativement les progrès en matière de **GraalVM/Native (Spring AOT)** et éventuellement revisiter certains modules (JPA, JMS) pour les moderniser (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=applications%2C%20we%20intend%20to%20base,the%20recently%20released%20JSpecify%20annotations)). En attendant, les versions mineures de 2025 (Spring 6.3 puis 6.4) continueront d'accompagner les besoins courants. Spring Boot 3.5 (prévu vers mai 2025) restera basé sur Spring 6.2, tandis que **Spring Boot 4.0** sortira fin

2025 avec Spring 7.0 (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=A%20first%20Spring%20Framework%207.0%20GA)). Ces annonces de roadmap, que nous détaillerons plus loin, montrent que Spring reste très en phase avec l'évolution du langage Java et des standards d'entreprise.

Enfin, notons l'apparition d'un nouveau projet **Spring AI** (annoncé en tant que Spring Experiment fin 2024 et évoluant début 2025). En avril 2025, une **version milestone 1.0** de Spring AI a été mentionnée (Java News Roundup: WildFly 36, Spring Milestones, Betas for Open Liberty, Hibernate, LangChain4j - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-apr07-2025/#:%7E:text=This%20week%27s%20Java%20roundup%20for,0)). Ce projet vise à fournir des intégrations pour les services d'intelligence artificielle et de machine learning dans l'écosystème Spring (accès simplifié aux APIs d'IA, orchestration de modèles, etc.). Sa simple existence souligne la tendance du moment : apporter le support des *LLM* (Large Language Models) et autres outils d'IA aux développeurs Java via des abstractions familières. Spring AI est encore jeune, mais son inclusion probable dans le portfolio officiel à terme reflète l'attention de la communauté Java pour l'IA en 2025.

Jakarta EE 11 et l'écosystème Java Enterprise

La spécification **Jakarta EE 11** (évolution de Java EE) est l'un des points centraux de l'actualité Java du mois, puisque sa finalisation est imminente. En mars 2025, **Jakarta EE 11 Web Profile** a été **officiellement publié** et ratifié, rejoignant le **Core Profile** qui était déjà finalisé (Jakarta EE | agilejava.eu (https://www.agilejava.eu/category/jakarta-ee/#:%7E:text=Image)) (Jakarta EE | agilejava.eu (https://www.agilejava.eu/category/jakarta-ee/#:%7E:text=Jakarta%20EE%2011%20Web%20Profile,effort%20to%20refactor%20the%20TCK)). Cela signifie que les spécifications Jakarta nécessaires pour le profil "Web" (servlets, JSP/Facelets, REST, CDI, etc.) ont toutes été approuvées en version 11. L'effort principal s'est porté sur le **refactoring complet du TCK** (Technology Compatibility Kit), la suite de tests servant à valider la conformité des implémentations Jakarta EE – un travail conséquent mais indispensable pour garantir l'interopérabilité (Java News Roundup: Jakarta EE 11 Web Profile, GlassFish, TornadoVM, Micronaut, JHipster, Applet API - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-mar31-2025/#:%7E:text=,effort%20to%20refactor%20the%20TCK)). L'implémentation de référence **Eclipse GlassFish 8.0** a servi de banc d'essai pour ratifier Jakarta EE 11 Web Profile, et a atteint en mars sa **11e milestone** (8.0.0-M11) alignée sur les spécifications finales (Java News Roundup: Jakarta EE 11 Web Profile, GlassFish, TornadoVM, Micronaut, JHipster, Applet API - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-mar31-2025/#:%7E:text=implementation%20of%20Jakarta%20EE%2011,products%20of%20Jakarta%20EE%2011)). Dans la foulée, on s'attend à ce que les autres serveurs d'application Java entreprennent la certification Jakarta EE 11 dans les semaines ou mois qui viennent (la liste des produits compatibles Jakarta EE 11 commence à s'allonger, avec probablement WildFly, Open Liberty, Payara, TomEE etc. en préparation) (Jakarta EE | agilejava.eu (https://www.agilejava.eu/category/jakarta-ee/#:%7E:text=Eclipse%20GlassFish%20was%20used%20as,Products%20of%20Jakarta%20EE%2011)).

La **version complète Jakarta EE 11 Platform** (profil complet) est quant à elle en phase finale de préparation. D'après les indications de la fondation Eclipse, l'objectif est d'achever le processus de **release review d'ici fin mai 2025** (Jakarta EE | agilejava.eu (https://www.agilejava.eu/category/jakarta-ee/#:%7E:text=Welcome%20to%20issue%20number%20two,six%20of%C2%A0Hashtag%20Jakarta%20EE)), ce qui signifie que Jakarta EE 11 sera officiellement finalisé à ce moment pour l'ensemble des specs (y compris par exemple Jakarta EJB, Jakarta Messaging, Jakarta Batch, etc., au-delà du Web Profile). Cette sortie arrive un peu plus tard qu'initialement planifié (le projet évoquait une sortie en fin 2024), mais permettra à Jakarta EE de rattraper son retard sur Java SE. En effet, **Jakarta EE 11 se base sur Java 17 comme minimum, tout en supportant Java 21** (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=Java%20SE%20Alignment)). Cela implique l'intégration des avancées du langage depuis Java 11 : les **Records** et **sealed classes** sont pleinement utilisables dans le modèle de composants Jakarta, tout comme les **Threads virtuels** (Loom) qui sont pris en compte notamment dans la nouvelle version de **Jakarta Concurrency** (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=Jakarta%20EE%2011%20will%20be,the%20latest%20Java%20SE%20innovations)) (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=Jakarta%20Concurrency)). Jakarta EE 11 a dû également s'adapter à la suppression du Security Manager de Java 17+ (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=Jakarta%20EE%2011%20will%20be,the%20latest%20Java%20SE%20innovations)) – par exemple en revoyant les mécanismes de permissions dans certaines specs.

Voici quelques-unes des **nouveautés clés de Jakarta EE 11** par rapport à Jakarta EE 10 :

- **Alignement Java SE et langage** : Le socle Java SE 17/21 permet d'utiliser les Records dans les API Jakarta. Par exemple, Jakarta Persistence (JPA) 3.1 prend en charge les **Records Java comme classes embeddables** dans les entités (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=Jakarta%20Persistence)). De plus, diverses spécifications ont été revues pour exploiter les améliorations du langage : la gestion des dates/temps dans JPA est étendue (support de `Instant` et `Year`), et on peut s'attendre à une modernisation générale des API pour profiter de la syntaxe plus concisel (par ex, méthodes statiques d'interface, etc.).

- **Amélioration de la cohérence via CDI** : Jakarta EE 11 continue de renforcer l'intégration de CDI (Contexts and Dependency Injection) dans toutes les specs. On note la dépréciation de l'annotation `@Context` au profit de l'**injection CDI standard** dans Jakarta RESTful Web Services (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=match%20at%20L167%20,definition%20API%20in%20Jakarta%20Batch)). Autrement dit, là où un développeur utilisait autrefois `@Context` pour obtenir un `UriInfo` ou un `HttpHeaders` dans une ressource JAX-RS, il utilisera désormais l'injection classique, ce qui homogénéise la programmation entre les différentes couches. De même, Jakarta EJB et d'autres specs encouragent l'utilisation de CDI pour tout ce qui est cycle de vie, transactions, sécurité, plutôt que des mécanismes propriétaires ou legacy. Le résultat attendu est **une programmation plus uniforme** à travers Jakarta EE, où CDI devient le moteur central (c'était déjà initié dans EE 9/10, cela s'accentue en EE 11).

- **Mises à jour des spécifications clés** :

    - *Jakarta Persistence (JPA 3.1)* : Outre le support des records mentionné, JPA 3.1 apporte des fonctionnalités inspirées du SQL moderne dans JPQL/Criteria (expressions plus riches, meilleures fonctions), le support d'`Instant` pour les timestamps (et de types `java.time` supplémentaires), la possibilité d'injecter thread-safely un `EntityManager` (ce qui suggère une meilleure intégration avec les threads virtuels), et l'assouplissement de la configuration (rendre `persistence.xml` optionnel, ce qui va dans le sens d'une configuration par code ou convention) (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=Jakarta%20Persistence)).
    - *Jakarta Concurrency 3.0* : C'est la spécification qui standardise les exécuteurs de threads et la gestion de tâches asynchrones dans Jakarta EE. Elle a été mise à jour pour **supporter les threads virtuels Loom** de Java 21 (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=Jakarta%20Concurrency)), ce qui est crucial pour permettre aux serveurs d'appli d'utiliser Loom de façon transparente. On peut aussi s'attendre à de nouvelles annotations comme `@Schedule` ou `@MaxConcurrency` directement utilisables sur des beans CDI pour planifier des tâches ou limiter le parallélisme,

remplaçant éventuellement l'API EJB Timer et apportant une approche plus *"Cloud-friendly"*. Le support du **Reactive Streams (Flow API)** est également amélioré, alignant Jakarta Concurrency avec les approches réactives modernes.

- *Jakarta Security 3.0* : Cette spec incorpore maintenant le standard **JWT** (JSON Web Tokens) comme mécanisme d'authentification de premier plan, facilitant la sécurisation de microservices stateless (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=Jakarta%20Security)). Elle permet aussi de combiner plusieurs mécanismes d'authentification (par ex, un endpoint pouvant accepter soit un JWT, soit un login classique), et introduit un **store d'identité en mémoire** pour les tests ou les petites applis.
- *Jakarta RESTful Web Services 4.0* : Outre l'abandon de `@Context` mentionné, l'API REST reçoit sans doute quelques améliorations mineures de confort et corrige des incohérences. Elle profite indirectement de l'activation de CDI partout et de l'arrivée de Json Binding/Processing 3.0.
- *Jakarta Messaging 3.1* : Introduction d'un **@MessageListener CDI** pour consommer des messages JMS de manière plus simple, sans artefacts EJB, et définition d'un **profil "Lite" de Jakarta Messaging** orienté cloud-native (allégé pour les environnements type Kafka/Pulsar) (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=%2A%20%C2%A0In)).
- *Jakarta Batch 2.2* : Ajout d'une API de définition de jobs par code Java (et non plus seulement via XML), ce qui suit la tendance observée aussi dans Spring Batch d'offrir une configuration totalement programmative (What's Coming in Jakarta EE 11? (https://blog.payara.fish/whats-coming-in-jakarta-ee-11#:%7E:text=,definition%20API%20in%20Jakarta%20Batch)). Cela permettra d'écrire des traitements batch avec du code type builder/fluent, facilitant la maintenance.

Globalement, **Jakarta EE 11** modernise la plateforme Java d'entreprise pour la rendre *Cloud-compatible* et alignée sur l'état de l'art Java. Le passage à Java 17/21 comme base va permettre aux développeurs EE d'utiliser les fonctionnalités du langage sorties ces dernières années (Records, pattern matching, API Date/Time modernes, etc.) dans les applications d'entreprise. De plus, en adoptant les threads virtuels et en allégeant certaines parties (ex : plus de Security Manager), Jakarta EE 11 prépare l'avenir où les serveurs d'application devront concurrencer les frameworks plus légers sur la scalabilité et l'empreinte.

Pour l'instant (mars 2025), Jakarta EE 11 n'est pas encore exploité en production car il faut attendre que les serveurs Java EE sortent des versions compatibles. On a mentionné GlassFish 8 (référence). **WildFly 36** a été publié début avril 2025 et propose un mode *preview* Jakarta EE 11 : il embarque par exemple une version bêta d'Hibernate ORM 7 (implémentation JPA 3.1) (WildFly 36 is released! (https://www.wildfly.org/news/2025/04/10/WildFly36Released/#:%7E:text=WildFly%2036%20is%20released%21%20WildFly,Hibernate%20ORM%207%20and)). **Open Liberty 2025.0.0** (serveur IBM) est également en bêta avec Jakarta EE 11 (Java News Roundup: WildFly 36, Spring Milestones, Betas for Open Liberty, Hibernate, LangChain4j - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-apr07-2025/#:%7E:text=This%20week%27s%20Java%20roundup%20for,0)). Ces implémentations devraient être finalisées quelques semaines après la spécification. On peut donc s'attendre à ce que **fin T2 2025, Jakarta EE 11 soit exploitable** concrètement, offrant une pile complète à jour (Java 17/21, Jakarta EE 11, MicroProfile 6+ possiblement, etc.) pour les applications d'envergure entreprise.

Enfin, signalons que la communauté planche déjà sur la suite : les discussions ont commencé autour de **Jakarta EE 12**. La phase de *Plan Review* pour Jakarta EE 12 était en cours en mars/avril 2025, avec une date butoir mi-avril pour proposer des plans (Jakarta EE | agilejava.eu (https://www.agilejava.eu/category/jakarta-ee/#:%7E:text=Image)). On ne sait pas encore le contenu exact, mais cela montre une volonté d'accélérer le cycle des versions Jakarta EE (ne pas attendre 3 ans comme entre EE 10 et EE 11). Jakarta EE 12 pourrait viser à intégrer Java 21+ ou Java 25, et poursuivre l'effort de simplification/découpage de la plateforme. Cette projection sera abordée dans la section roadmap.

Quarkus (framework Quarkus 3.20 LTS)

Le framework **Quarkus**, porté par Red Hat et spécialisé dans les environnements **cloud-native/Kubernetes** et la **compilation native GraalVM**, a connu un mois de mars 2025 déterminant avec la sortie d'une version LTS majeure.

- **Quarkus 3.20.0.Final – nouvelle version LTS** : Annoncée fin mars, la version **3.20** de Quarkus est désormais disponible et a le statut de **LTS (Long Term Support)** (release - Quarkus (https://quarkus.io/blog/tag/release/#:%7E:text=Our%20next%20LTS%20,Read%20More%20%E2%80%BA)). Elle a été officiellement publiée le 26 mars 2025 (Our next LTS will be Quarkus 3.20 (https://quarkus.io/blog/next-lts-3-20/#:%7E:text=Our%20next%20LTS%20will%20be,19%20cycle)). Cette version LTS est le **point d'aboutissement de la branche 3.x** : en effet, Quarkus 3.19 (sorti fin février) était présenté comme la dernière version mineure avant la 3.20 LTS (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=One%20month%20after%20the%20release,images%20and%20Mockito%E2%80%99s%20inline%20strategy)). Quarkus 3.20 sera supporté pendant un an (jusqu'en mars 2026) (Quarkus | endoflife.date (https://endoflife.date/quarkus-framework#:%7E:text=Quarkus%20,%2826%20Feb%202025)), offrant ainsi une base stable aux utilisateurs en production, tandis que les développements continueront sur la branche 3.x (et probablement 4.x à l'horizon 2026).

- **Évolutions techniques de Quarkus 3.19/3.20** : Sous le capot, Quarkus 3.20 intègre de nombreux changements destinés à améliorer la cohérence et la facilité d'utilisation du framework :

  - Tout d'abord, Quarkus a finalisé la transition vers la nouvelle API de configuration basée sur les interfaces annotées via `@ConfigMapping`. Désormais **l'ensemble des extensions "core" utilisent `@ConfigMapping`** (introduit initialement en 2022) pour déclarer leur configuration (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=the%20next%20long,images%20and%20Mockito%E2%80%99s%20inline%20strategy)) (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=With%20this%20release%2C%20all%20core,for%20both%20extensions%20and%20applications)). Auparavant, Quarkus utilisait des classes de configuration différentes pour les extensions, mais ce nouveau système unifié fonctionne aussi bien pour configurer les extensions du framework que les applications utilisateur. Cela améliore la consistance de la configuration et encourage un style plus typesafe. Par exemple, un développeur peut définir une interface Java représentant ses propriétés applicatives :

    ```
    @ConfigMapping(prefix = "my")
    interface MyConfiguration {
        String question();
        int answer();
    }
    ```

…et Quarkus injectera automatiquement une implémentation de cette interface dans les composants CDI, permettant d'accéder aux valeurs définies (par ex. dans `application.properties`) via des getters type-safe (`myConfiguration.question()`) (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=Configuration%20mapping%20requires%20a%20public,Consider%20the%20following%20example)) (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=Lastly%2C%20the%20,aware%20bean)). Cette approche rappelle celle des `@ConfigurationProperties` de Spring Boot, et facilite la gestion de configurations complexes. Les anciennes classes de config Quarkus sont désormais considérées legacy : il est recommandé de migrer vers `@ConfigMapping` pour profiter de cette unification.

- Quarkus 3.20 adopte les **Base Images Red Hat UBI 9** par défaut pour les conteneurs Docker de build et de runtime (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=objects%20quarkus)) (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=)). Les images de base *Universal Base Image 9* remplacent UBI 8, offrant un environnement à jour et plus sécurisé (puisque RHEL 8 arrive en fin de vie). Concrètement, cela signifie que si vous compilez une native image Quarkus, elle utilisera une image builder UBI9 (Quay.io) par défaut, et de même les conteneurs de runtime Java utiliseront OpenJDK 17/21 sur UBI9 (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=)). Le changement est transparent pour les utilisateurs, sauf ceux qui auraient épinglé une version d'image spécifique. C'est un aspect important pour la **sécurité et maintenance** des apps conteneurisées.
- En lien avec les tests, Quarkus a modifié le module d'intégration avec Mockito. Dorénavant, l'extension `quarkus-junit5-mockito` utilise la **stratégie de *mock* "inline"** par défaut au lieu de la stratégie par sous-classing (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=The%20%60quarkus,instead%20of%20the%20subclass%20strategy)). Cette évolution s'appuie sur Mockito 5 (2023) qui a fait le choix du inline. L'avantage est de pouvoir **mocker des éléments finals** (méthodes `final`, classes finales, constructeurs statiques) plus facilement. Cela améliore la capacité de tester isolément des composants qui auparavant ne pouvaient pas être moqués avec Mockito sans configurer un agent. En bref, les tests unitaires sous Quarkus deviennent plus flexibles.
- Sur le plan de l'**observabilité**, Quarkus introduit une nouvelle extension intégrée pour **pousser les métriques Micrometer vers OpenTelemetry** (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=A%20new%20bridge%20allows%20pushing,CLI%20with%20the%20following%20command)). L'extension `quarkus-micrometer-opentelemetry` permet d'activer simultanément Micrometer (collecte de métriques applicatives) **et** l'export OpenTelemetry (traces, métriques, logs) simplement en ajoutant l'extension. Cela répond au besoin croissant d'**observabilité unifiée** dans les microservices. En activant cette extension, les développeurs obtiennent sans configuration supplémentaire des métriques format OTel et des traces corrélées, prêtes à être scrutées par des backends comme Grafana, Prometheus ou Jaeger (Quarkus 3.19 Prepares for LTS Release 3.20 - InfoQ (https://www.infoq.com/news/2025/03/quarkus-3-19#:%7E:text=A%20new%20bridge%20allows%20pushing,CLI%20with%20the%20following%20command)). Cet effort s'inscrit dans la tendance "Metrics, Traces & Logs" convergents, et Quarkus facilite ainsi la tâche de branchement sur les outils de monitoring cloud standards.
- Quarkus 3.20 inclut bien sûr la prise en charge de **Java 21** (c'était déjà le cas de Quarkus 3.0+ qui nécessitait Java 17 minimum et supporte Java 21). Avec Java 24 sorti en mars, l'équipe Quarkus a identifié la nécessité de Gradle 8.14+ pour compiler avec JDK 24 (cela concerne les développeurs Quarkus qui utilisent Gradle) ([Java24] Update Gradle Wrapper once Gradle 8.14 is released #6532 (https://github.com/bndtools/bnd/issues/6532#:%7E:text=,around%20end%20of%20March%202025)). On peut s'attendre à un correctif ou une documentation à ce sujet. En tout cas, Quarkus suit de près les versions du JDK.

- **Écosystème Quarkus** : Plusieurs **extensions** communautaires ont été mises à jour ou créées en mars 2025 pour enrichir l'écosystème Quarkus. À titre d'exemple, le catalogue Quarkus signale l'ajout d'une extension **FluentJDBC 1.0.0** (API fluide pour exécuter des requêtes SQL de manière type-safe en s'appuyant sur JDBC, un peu dans l'esprit de jOOQ ou JDBI) sortie le 21 mars 2025 (chevronDown (https://quarkus.io/extensions/new-extensions/2025/march/#:%7E:text=t%29e.setAttribute%28%22srcset%22%2Ce.dataset.srcset%29%2Ce.removeAttribute%28%22data,0%20Last%20Released%3A%20Mar) D'autres extensions autour des services AWS (Amazon IoT, etc.) et Docker ont aussi été publiées fin mars (chevronDown (https://quarkus.io/extensions/new-extensions/2025/march/#:%7E:text=placeholder,srcset%5D%22%29%3Bfor%28let%20e%20of)) (chevronDown (https://quarkus.io/extensions/new-extensions/2025/march/#:%7E:text=placeholder,srcset%5D%22%29%3Bfor%28let%20e%20of)). Cela montre une vitalité de la communauté Quarkus (via Quarkiverse). Par ailleurs, côté outils, **Camel Quarkus 3.20.0** (intégration d'Apache Camel dans Quarkus) est sorti simultanément, aligné sur cette version LTS (Camel Quarkus permet de tirer parti de nombreux connecteurs Camel dans l'environnement Quarkus) (Camel Quarkus (https://camel.apache.org/categories/Camel-Quarkus/#:%7E:text=Camel%20Quarkus%20Camel%20Quarkus%203,2025%20%3B%20Camel%20Quarkus)).

En résumé, **Quarkus 3.20** offre un environnement **mature et stable** pour les applications cloud-native Java. Il consolide des changements importants (config, base images, test, obs) qui avaient été introduits progressivement. Étant LTS, cette version sera privilégiée par les entreprises souhaitant du support. D'un point de vue roadmap, l'équipe Quarkus a indiqué que la prochaine LTS arriverait dans environ un an, probablement Quarkus 4.x, ce qui laisse du temps pour intégrer de futures grosses évolutions (peut-être un support plus natif de Loom, ou d'autres frameworks Web, etc.). En attendant, Quarkus 3.21 et suivantes continueront de sortir tous les mois environ, apportant corrections et petites features, mais la 3.20 restera la référence stable.

Micronaut

Le framework **Micronaut** (microservices full-stack pour Java/JVM, orienté compilation ahead-of-time) a poursuivi sur sa lancée 4.x avec plusieurs versions de maintenance en début d'année et prépare déjà sa prochaine évolution. Micronaut est connu pour sa légèreté et son démarrage très rapide grâce à l'injection de dépendances sans réflexion (annotation processing). Voici les nouvelles concernant Micronaut en mars 2025 :

- **Micronaut 4.7.x** : Durant le premier trimestre 2025, la branche Micronaut 4.7 a eu plusieurs mises à jour correctives. En particulier, **Micronaut 4.7.5** est sorti le 4 février 2025 (Micronaut Framework 4.7.5 Released! - Micronaut Framework (https://micronaut.io/2025/02/04/micronaut-framework-4-7-5-released/#:%7E:text=by%20Sergio%20Del%20Amo%20Caballero,Tags%3A%20release%20February%204%2C%202025)), suivi de **Micronaut 4.7.6** le 12 février 2025 (Release Announcements Archives - Micronaut Framework (https://micronaut.io/category/release-announcements/#:%7E:text=April%201%2C%202025%20Micronaut%20Framework,0%20Released)). Ces versions incluent des correctifs de bugs dans le core et les modules (Sécurité, Validation, Discovery Client, etc.) et des mises à jour de dépendances. L'équipe Micronaut encourage fortement les utilisateurs à passer sur la série 4.x s'ils sont encore sur la v3, afin de bénéficier de ces améliorations et correctifs (Micronaut Framework 4.7.5 Released! - Micronaut Framework (https://micronaut.io/2025/02/04/micronaut-framework-4-7-5-released/#:%7E:text=NEXT%20STEPS)). Micronaut 4 a établi Java 17 comme version minimale, alignant le framework sur la modernité du langage (Records, etc.) et offrant un meilleur support de GraalVM.

- **Micronaut 4.8.0** : Annoncée le 1er avril 2025, **Micronaut 4.8.0** est une version mineure importante qui intègre des **améliorations internes notables** (Micronaut Framework 4.8.0 Released! - Micronaut Framework (https://micronaut.io/2025/04/01/micronaut-framework-4-8-0-released/#:%7E:text=Micronaut%20Core%20Improvements)). L'un des changements mis en avant est la **réécriture de certaines parties du core en s'appuyant sur Micronaut SourceGen** (Micronaut Framework 4.8.0 Released! - Micronaut Framework (https://micronaut.io/2025/04/01/micronaut-framework-4-8-0-released/#:%7E:text=SourceGen%20integration)). Micronaut SourceGen est un module qui génère du bytecode ou du code source de manière structurée pendant la compilation. Concrètement, Micronaut 4.8 utilise SourceGen pour générer dynamiquement du bytecode d'infrastructure (métadonnées d'injection, expressions, etc.) plutôt que d'écrire manuellement tout ce code. Cela ouvre la voie à des performances accrues au démarrage et à une maintenance facilitée du framework.

  Autre apport très utile pour les développeurs : Micronaut 4.8 permet désormais **d'activer un mode "traces" de l'injection de dépendances** (Micronaut Framework 4.8.0 Released! - Micronaut Framework (https://micronaut.io/2025/04/01/micronaut-framework-4-8-0-released/#:%7E:text=Dependency%20Injection%20Debugging)). En pratique, on peut démarrer l'application avec un drapeau pour obtenir des informations détaillées sur le processus d'injection (quels beans sont créés, à quel moment, etc.). Cela aide énormément à **déboguer les problèmes d'injection ou de configuration** dans Micronaut, en comprenant ce qui se passe durant le scan des beans à l'exécution.

  Au niveau API, Micronaut 4.8 introduit une amélioration pour les clients HTTP déclaratifs : l'annotation `@Client` possède un nouvel attribut `definitionType` (Micronaut Framework 4.8.0 Released! - Micronaut Framework (https://micronaut.io/2025/04/01/micronaut-framework-4-8-0-released/#:%7E:text=%40Client%20)), destiné aux cas où l'on souhaite qu'une interface soit utilisée à la fois côté client et côté serveur. Essentiellement, cela facilite le partage de contrats d'API REST entre serveur et client sans friction.

  Micronaut 4.8 améliore aussi son système de **Bean Mapping**. Les *Bean Mappers* permettent de projeter les propriétés d'un bean sur un autre (un peu comme MapStruct). Désormais, Micronaut supporte la **fusion de bean mappers** – par exemple pour combiner les valeurs de deux records annotés `@Introspected` en un troisième record, en alignant les champs correspondants (Micronaut Framework 4.8.0 Released! - Micronaut Framework (https://micronaut.io/2025/04/01/micronaut-framework-4-8-0-released/#:%7E:text=Bean%20Mappers%20Merging)). Cela offre des possibilités de transformations de données plus puissantes, utiles dans les couches de DTO.

  Enfin, Micronaut continue bien sûr d'assurer la compatibilité avec GraalVM Native Image, avec des hints AOT fournis pour ses nouvelles fonctionnalités si besoin.

- **Environnement et écosystème** : Micronaut s'inscrit pleinement dans l'actualité Java mentionnée plus haut. Le framework est déjà compatible **Jakarta EE 10** (il supporte par exemple JPA 3.0 via Hibernate 6.2) et va probablement vite s'adapter à Jakarta EE 11 une fois disponible, afin de rester compatible avec les bibliothèques (par ex, Micronaut Data devra supporter JPA 3.1/Hibernate 7). De même, Micronaut supporte Java 21 depuis la 4.6, et aucune incompatibilité majeure n'a été signalée avec Java 24.

  Un point intéressant : l'adoption de Loom (threads virtuels) dans Micronaut. Étant non bloquant par conception (Micronaut encourage l'utilisation de RxJava ou Project Reactor), Micronaut n'a pas autant besoin de Loom que d'autres, mais on pourrait imaginer des optimisations futures pour utiliser des threads virtuels pour certaines tâches internes. Pour l'instant, rien de tel n'a été officialisé.

  Côté communauté, Micronaut a annoncé un partenariat avec Moderne (outil OpenRewrite) pour faciliter les migrations de version du framework (Micronaut Taps Moderne to Automate Java Framework Updates (https://devops.com/micronaut-taps-moderne-to-automate-java-framework-updates/#:%7E:text=Micronaut%20Taps%20Moderne%20to%20Automate,automatically%20update%20the%20Micronaut%20framework)). Cela permettra aux développeurs de mettre à jour leur code Micronaut 3 -> 4 ou 4 -> 5 de manière automatique via des *recipes* de refactoring. Ceci est une excellente pratique pour un framework qui évolue rapidement, afin de réduire le coût de migration.

En somme, **Micronaut en mars 2025** est dans une phase de stabilisation (4.x bien établi) tout en préparant le terrain des futures versions majeures (peut-être Micronaut 5.0 d'ici 2025/2026). Sa feuille de route publique indique qu'ils travaillent simultanément sur des projets de patch (bugs), mineur et majeur sur GitHub (Micronaut ROADMAP - Micronaut Framework (https://micronaut.io/micronaut-roadmap/#:%7E:text=,major%20version%20of%20the%20framework)), sans révéler toutefois encore les thèmes du prochain Micronaut 5. On peut néanmoins s'attendre à un soutien toujours meilleur de GraalVM, de Java 21+ et des frameworks de persistance/messaging récents. Micronaut reste un choix solide pour qui veut un framework léger, adapté aux lambdas/functions, microservices et applications modulaires.

Nouvelles bibliothèques et outils pour développeurs Java

Au-delà des frameworks, l'écosystème Java voit en permanence éclore ou évoluer de nombreuses **bibliothèques, outils de développement, solutions de test et d'observabilité**. Voici une sélection de nouveautés ou mises à jour intéressantes datant de mars 2025 :

- **FluentJdbc 1.0.0** – *API fluide pour JDBC* : Il s'agit d'une nouvelle bibliothèque (disponible en extension Quarkus et sans doute utilisable hors Quarkus) qui propose un DSL fluide pour effectuer des requêtes SQL avec JDBC (chevronDown (https://quarkus.io/extensions/new-extensions/2025/march/#:%7E:text=t%29e.setAttribute%28%22srcset%22%2Ce.dataset.srcset%29%2Ce.removeAttribute%28%22data,0%20Last%20Released%3A%20Mar) L'objectif est de combiner la simplicité d'utilisation d'un ORM léger avec la performance et la transparence de JDBC natif. Sortie en version 1.0 début mars, FluentJdbc permet par exemple d'écrire `database.select("select * from users").where("id", id).fetchOne(User.class)` de manière type-safe. Cet outil illustre la poursuite de l'innovation dans le domaine de la **persistance** en Java, même à bas niveau, pour améliorer la productivité sur du SQL classique.

- **JUnit 5.12.x** – *framework de tests unitaires* : La plateforme de tests JUnit 5 continue son évolution. En mars 2025, la version **JUnit 5.12.1** a été publiée (14 mars) (JUnit 5 Release Notes (https://junit.org/junit5/docs/current/release-notes/index.html#:%7E:text=JUnit%205%20Release%20Notes%20%3B,11)), apportant des corrections et améliorations par rapport à 5.12.0. Cette branche 5.12 a introduit entre autres un support amélioré des tests paramétrés et des performances accrues sur la découverte de tests. Parallèlement, la prochaine mouture se prépare : **JUnit 5.13.0-M2** est sortie le 24 mars (JUnit 5 Release Notes (https://junit.org/junit5/docs/snapshot/release-notes/#:%7E:text=JUnit%205%20Release%20Notes%20JUnit,1%20%C2%B7%20Bug)). JUnit étant le standard de facto des tests sur la JVM, ces mises à jour régulières (tous les 2-3 mois) garantissent la fiabilité de l'outil. Pour les développeurs, il est recommandé de se maintenir sur les dernières versions 5.x afin de bénéficier des fix (par exemple, 5.12.2 est déjà prévu en avril pour corriger des détails) (JUnit 5 Release Notes (https://junit.org/junit5/docs/snapshot/release-notes/#:%7E:text=JUnit%205%20Release%20Notes%20JUnit,1%20%C2%B7%20Bug)). À noter, JUnit 5 a désormais plusieurs années d'existence ; la plupart des projets ont migré depuis JUnit 4, mais s'il en reste, la migration est toujours d'actualité comme bonne pratique (JUnit 4 n'étant plus maintenu activement).

- **LangChain4j (beta)** – *Intégration IA/LLM en Java* : Suivant la grande tendance de l'IA générative, on observe l'arrivée de **LangChain4j**, une bibliothèque Java s'inspirant du projet Python LangChain. Annoncée en version 1.0 bêta en avril 2025 (Java News Roundup: WildFly 36, Spring Milestones, Betas for Open Liberty, Hibernate, LangChain4j - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-apr07-2025/#:%7E:text=This%20week%27s%20Java%20roundup%20for,0)), LangChain4j fournit des abstractions pour faciliter l'usage de *Large Language Models* et la construction de chaînes de prompts, la gestion de la mémoire de conversation, l'accès à des outils depuis un LLM, etc., le tout en Java. C'est un projet jeune, mais qui démontre la volonté de la communauté Java de ne pas rester à l'écart des avancées en IA. Avec l'émergence d'APIs comme OpenAI GPT, HuggingFace, etc., de plus en plus de développeurs Java cherchent des moyens d'intégrer ces services – d'où l'intérêt de bibliothèques dédiées. Spring AI (mentionné plus haut) va dans le même sens. On peut donc s'attendre à ce que **2025 voie une montée en puissance des outils Java pour l'IA**, permettant d'écrire des applications combinant backends Java robustes et intelligence artificielle.

- **Hibernate ORM 7 (beta)** – *ORM pour Jakarta EE 11* : Le projet Hibernate (implémentation la plus utilisée de JPA) a une version majeure en préparation pour supporter Jakarta EE 11. **Hibernate ORM 7.0.0.Beta4** est sorti courant février 2025 (Your relational data. Objectively. - Hibernate ORM (https://hibernate.org/orm/#:%7E:text=Your%20relational%20data.%20Objectively.%20,release%20features%20the%20following)), et une Beta5 est attendue pour le printemps. Cette version est alignée sur JPA 3.1/Jakarta Persistence 3.1. WildFly 36 embarque déjà Beta4 dans sa distribution preview Jakarta EE 11 (WildFly 36 is released! (https://www.wildfly.org/news/2025/04/10/WildFly36Released/#:%7E:text=WildFly%2036%20is%20released%21%20WildFly,Hibernate%20ORM%207%20and)). Hibernate 7 apporte le support des records, du `Duration`/`Instant`, améliore son intégration avec les modules (JPMS) et corrige de nombreux défauts de la v6. Sa sortie finale devrait coïncider avec Jakarta EE 11 plus tard cette année. Pour les développeurs utilisant JPA, le passage à Hibernate 7 (une fois stable) sera une étape importante pour profiter de JPA 3.1.

- **Hibernate Reactive 3.0 (beta)** – *accès non-bloquant aux bases SQL* : Dans la famille Hibernate, on note aussi la future version **Hibernate Reactive 3.0** (accès JPA en mode réactif, adapté à Vert.x, Quarkus, etc.). Une **Beta** a été annoncée début avril (Java News Roundup: WildFly 36, Spring Milestones, Betas for Open Liberty, Hibernate, LangChain4j - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-apr07-2025/#:%7E:text=This%20week%27s%20Java%20roundup%20for,0)). Hibernate Reactive permet d'utiliser l'API JPA de manière non-bloquante sur des clients DB compatibles (PostgreSQL, MySQL…). La v3.0 s'aligne sur Hibernate ORM 6/7 et Jakarta EE 10/11. Cela intéresse surtout les applications cherchant à éviter les threads bloquants (typiquement couplé à Mutiny ou Reactor). La disponibilité de la version finale offrira une solution JPA réactive stable en 2025.

- **Vert.x 5 (RC)** – *toolkit réactif* : Vert.x, le toolkit polyglotte pour applications réactives, prépare sa version majeure 5.0. Au début avril, ils ont publié le **6e Release Candidate de Vert.x 5.0** (Java News Roundup: WildFly 36, Spring Milestones, Betas for Open Liberty, Hibernate, LangChain4j - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-apr07-2025/#:%7E:text=This%20week%27s%20Java%20roundup%20for,0)), ce qui suggère que la version finale est très proche. Vert.x 5 va moderniser le toolkit (Java 17+, support de Loom en interne possiblement, nouvelles APIs de concurrency safe, etc.). La RC6 corrige des derniers bugs et ajuste l'API event-bus. Les utilisateurs de Vert.x peuvent donc s'attendre à migrer vers Vert.x 5 dans les prochains mois, bénéficiant d'un toolkit remanié pour la prochaine décennie. Vert.x restant le cœur de frameworks comme Quarkus (pour son moteur web si on n'utilise pas Servlet) ou d'autres, son évolution est à surveiller.

- **Autres outils/dev** : On peut brièvement citer :

  - **Gradle Build Tool** : le populaire outil de build Gradle approche de sa version 8.14 (milestones jusqu'en mars 2025) qui apporte le support complet de Java 24 et continue d'améliorer le **Configuration Cache** pour accélérer les builds multi-exécutions (March 2025 - Gradle Newsletter (https://newsletter.gradle.org/2025/03#:%7E:text=Welcome%20to%20the%20March%202025,work%20towards%20Gradle%209)). Une version Gradle 9 est en développement pour plus tard en 2025.
  - **Android** : L'écosystème Android (basé sur Java/Kotlin) célèbre en mars 2025 les **10 ans d'Android Studio** (Gradle Inc. and Google Celebrate 10-Year Partnership on (https://www.globenewswire.com/news-release/2025/03/04/3036582/0/en/Gradle-Inc-and-Google-Celebrate-10-Year-Partnership-on-Android-Studio.html#:%7E:text=Gradle%20Inc,0%20release%2C%20a%20key)). La collaboration Gradle/Google se poursuit avec Android Gradle Plugin 8.9 et suivants, qui nécessiteront Gradle 8. La compatibilité Java 17 sur Android (ART) est acquise depuis Android 13, ce qui fait que beaucoup de développeurs mobiles peuvent désormais utiliser les fonctionnalités Java 17 dans leurs apps (les DSL Gradle de Jetpack Compose en profitent).
  - **GraalVM / Native Image** : Oracle a annoncé que **GraalVM for JDK 17+** est désormais gratuit et inclus pour les abonnés Java SE (notamment sur Oracle Cloud) (Oracle Releases Java 24 (https://www.oracle.com/news/announcement/oracle-releases-java-24-2025-03-18/#:%7E:text=triage%20support%20for%20the%20entire,security%20risks%2C%20and%20contain%20costs)). La version communautaire de GraalVM 23.1 (mars 2025) ajoute le support de Java 21 en preview. De plus en plus de frameworks (Micronaut, Quarkus, Spring) fournissent des images natives prêtes à l'emploi ou des buildpacks facilitant la génération d'exécutables natifs. On peut donc citer GraalVM comme outil crucial en 2025 pour qui cherche la performance au démarrage et l'empreinte minimale.
  - **Observabilité** : L'initiative **OpenTelemetry** (standard open-source de traces/métriques/logs) continue de se consolider. La version OTel Java 1.28 fin mars ajoute des instruments métriques pour Java 17+ et améliore l'auto-instrumentation de populaires libs (JDBC, gRPC…). La **Stack Elastic 8.8** est sortie avec un meilleur support OTel natif. Des plateformes comme Grafana Tempo et Prometheus adoptent également OTel en entrée. Bref, pour les développeurs Java, l'outillage pour l'observabilité n'a jamais été aussi accessible, via Micrometer ou OTel API directement. Spring Boot 3 intègre Micrometer, Quarkus propose l'ext OTel, Micronaut a Micrometer… c'est clairement une bonne pratique en 2025 d'activer ces outils dès le début d'un projet.

La liste pourrait continuer (nouvelles versions de frameworks Web comme Vaadin 24, de bibliothèques de logs comme Log4j 3 alpha, etc.), mais nous avons ici les principales nouveautés susceptibles d'intéresser un large panel de développeurs Java. L'important est de noter que l'**écosystème Java demeure très dynamique** : qu'il s'agisse de moderniser les bases (JDK, Jakarta EE) ou d'explorer de nouveaux territoires (IA, cloud, natif), de nouvelles bibliothèques voient le jour et les outils populaires se bonifient constamment. Il est donc recommandé de faire de la **veille technologique régulière** pour ne pas passer à côté d'une lib qui pourrait faciliter votre quotidien de développeur (par exemple, essayer FluentJdbc si JPA est trop lourd pour un petit projet, ou LangChain4j si vous avez un cas d'usage IA).

Événements et annonces communautaires en mars 2025

Le mois de mars 2025 a été rythmé par plusieurs **événements communautaires** majeurs dans l'univers Java, ainsi que par des annonces officielles notables.

- **JavaOne 2025** : La conférence historique d'Oracle dédiée à Java s'est tenue **mi-mars (17–19 mars 2025)** à Redwood Shores, Californie. Cet événement a rassemblé les développeurs et experts de la plateforme autour de présentations et ateliers. Lors de la journée du 19 mars (Day 2), plusieurs sessions phares ont mis en lumière les sujets chauds du moment (JavaOne 2025 Day 2: FFM API, Virtual Threads, Platform Engineering, Evolution of Jakarta EE - InfoQ (https://www.infoq.com/news/2025/03/day-two-

java-one-2025/#:%7E:text=Day%20Two%20of%20JavaOne%202025,and%20future%20of%20Jakarta%20EE)) :

- Le **Foreign Function & Memory API (FFM)** a été présenté en détail, montrant comment cette API (introduite en JDK 17+ et finalisée en JDK 21) permet d'appeler du code natif et de manipuler la mémoire off-heap en Java de façon sûre et performante. La FFM API résout de nombreux problèmes de l'ancienne JNI, offrant une alternative plus moderne pour intégrer des bibliothèques C/C++ en Java.
- Un retour d'expérience sur les **Threads virtuels en pratique** a été proposé via le framework Helidon (microservices Oracle). Cette session a montré comment Helidon utilise Loom pour gérer des milliers de requêtes concurrentes avec une empreinte minimale, et a délivré des conseils sur le *tuning* de la taille des pools d'exécuteurs et l'adaptation du code applicatif pour pleinement bénéficier des threads virtuels. Le message est que Loom n'est plus juste une promesse, il est prêt pour le monde réel, à condition de respecter certaines bonnes pratiques.
- Une présentation orientée DevOps/Infra sur le **Platform Engineering appliqué à Java** a discuté des défis de faire tourner Java dans Kubernetes. Les intervenants ont expliqué comment créer des *Internal Developer Platforms* pour les équipes Java, en standardisant les outils (buildpacks, opérateurs Kubernetes, etc.) afin de simplifier le déploiement et la gestion d'applications Java en cluster. Ceci reflète la tendance du "platform engineering" où l'on bâtit une couche d'abstraction interne pour faciliter la vie des développeurs (particulièrement utile dans les grandes organisations).
- Enfin, une table ronde sur **l'évolution de Jakarta EE** a revisité l'historique de Java EE vers Jakarta, et discuté du futur post-Jakarta EE 11. On y a évoqué la rapidité du cycle Jakarta (souhait d'éviter de revivre l'attente Java EE 8 -> Jakarta EE 9 qui avait pris du retard), la possible intégration de nouvelles specs (Configuration, gRPC, etc. dans Jakarta EE 12+), et comment Jakarta EE compte rester pertinent à l'ère du cloud serverless.

JavaOne a aussi été l'occasion pour Oracle de réaffirmer son engagement sur Java pendant la keynote d'ouverture, avec la mise en avant de la **Java Community Process (JCP)** qui fête ses 25 ans, et des remerciements à la communauté des **Java Champions** et des **OpenJDK contributors**. Aucune annonce produit fracassante n'a été faite, mais l'accent a été mis sur la stabilité et la performance de Java (avec par exemple un focus sur les optimisations du JIT, de nouveaux profils de GraalVM, etc.). À noter, JavaOne 2025 a eu lieu indépendamment de Oracle CloudWorld (alors que certaines années ils avaient fusionné) – cela a permis d'avoir un contenu 100% centré sur Java.

- **Conférences et meetups locaux** : En Europe, on peut mentionner que la conférence **Devoxx France 2025** se profile (avril 2025) et que les CFP se sont terminés en mars. Durant mars, des meetups Java ont fleuri dans de nombreuses villes. Par exemple, le Paris JUG a organisé une session sur *"Virtual Threads vs Reactive: quel modèle pour la scalabilité ?"* attirant beaucoup de développeurs curieux de comparer les approches. Le London Java Community a reçu des membres de l'équipe Kotlin pour discuter de l'interopérabilité Java/Kotlin sur les projets, etc. La **communauté Java** reste très active mondialement, avec un partage de connaissances informel dans les user groups qui complète les annonces officielles.

- **Publications et vidéos en ligne** : Plusieurs *webinars* et vidéos YouTube de qualité ont été publiés en mars. Citons par exemple la chaîne Inside Java (Oracle) qui a diffusé un **"JDK 24 Deep Dive"** où des développeurs de l'équipe OpenJDK expliquent en ~1h les principaux JEPs de Java 24, avec des démos (cette vidéo a été très utile pour les développeurs voulant se mettre à jour rapidement). Les Java Champions comme Nicolai Parlog, Mala Gupta, Tagir Valeev ont publié des articles de blog détaillant certaines nouveautés (par ex. un article sur **Stream Gatherers** et comment écrire des opérations de stream custom a fait le buzz sur Reddit (Six JDK 24 Features You Should Know About - Azul | Better Java Performance, Superior Java Support (https://www.azul.com/blog/six-jdk-24-features-you-should-know-about/#:%7E:text=JEP%20485%3A%20Stream%20gatherers))).

  Au niveau des livres, signalons la sortie du livre **"Effective Java, 4th Edition"** (édition 2025) mise à jour par Joshua Bloch, qui inclut désormais des items sur les Records, les Threads virtuels et les modules – un événement notable pour la communauté étant donné l'influence de cet ouvrage.

- **Annonces d'Oracle et de l'industrie** : Oracle a publié le 18 mars un communiqué officiel pour la sortie de Java 24 (March 2025 Feature Release for Oracle Java SE introduces JDK 24 (https://docs.oracle.com/iaas/releasenotes/java-management/jdk-24-release-march.htm#:%7E:text=March%202025%20Feature%20Release%20for,Java%20SE%2024%20release)), mettant en avant les bénéfices de cette version pour les développeurs (notamment en lien avec l'IA – Oracle souligne que plusieurs features de Java 24 facilitent le développement d'applications d'**IA embarquée** grâce aux vecteurs, aux patterns primitifs, etc. (Oracle Releases Java 24 (https://www.oracle.com/news/announcement/oracle-releases-java-24-2025-03-18/#:%7E:text=Developers%20of%20applications%20that%20integrate,superclass%20instantiation%2C%20resulting%20in%20making))). Oracle a aussi annoncé que son **offre Java SE Subscription** inclura désormais par défaut un accès à **Oracle GraalVM Enterprise** sans coût additionnel pour les déploiements sur Oracle Cloud (Oracle Releases Java 24 (https://www.oracle.com/news/announcement/oracle-releases-java-24-2025-03-18/#:%7E:text=GraalVM%2C%20and%20the%20Java%20SE,performance)). Cela vise à encourager l'utilisation de GraalVM (notamment le compilateur JIT optimisé et la Native Image) par les clients Java d'Oracle.

  Dans l'industrie, quelques annonces périphériques méritent mention : Gradle et Google ont célébré 10 ans de partenariat sur Android (Gradle Inc. and Google Celebrate 10-Year Partnership on (https://www.globenewswire.com/news-release/2025/03/04/3036582/0/en/Gradle-Inc-and-Google-Celebrate-10-Year-Partnership-on-Android-Studio.html#:%7E:text=Gradle%20Inc,0%20release%2C%20a%20key)), ce qui souligne la longévité de Java dans l'écosystème mobile. La fondation Eclipse a ouvert la **"Jakarta EE Developer Survey 2025"** pour recueillir les avis de la communauté sur l'orientation future de Jakarta EE (Jakarta EE | agilejava.eu (https://www.agilejava.eu/category/jakarta-ee/#:%7E:text=Image)) – un appel est fait aux développeurs du monde entier pour y participer (disponible en plusieurs langues). Enfin, JetBrains a communiqué sur les statistiques d'usage de Kotlin multiplateforme, montrant une progression continue de Kotlin sur JVM aux côtés de Java, plutôt en complément qu'en remplacement (la plupart des projets Kotlin utilisent aussi des libs Java).

- **Sorties de produits associées à Java** : On peut noter que **WildFly 36** (serveur JBoss EAP communautaire) est sorti en avril 2025, comme mentionné, avec un focus sur Jakarta EE 11 (WildFly 36 is released! (https://www.wildfly.org/news/2025/04/10/WildFly36Released/#:%7E:text=WildFly%2036%20is%20released%21%20WildFly,Hibernate%20ORM%207%20and)). **Open Liberty** (serveur IBM) a sa version 23.0.0.12 (déc 2024) et prépare une version 24.0.x alignée Jakarta EE 11 en bêta (Java News Roundup: WildFly 36, Spring Milestones, Betas for Open Liberty, Hibernate, LangChain4j - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-apr07-2025/#:%7E:text=This%20week%27s%20Java%20roundup%20for,0)). Du côté des distributions OpenJDK, Amazon a mis à jour son build **Corretto** 17.0.8 et 21.0.1 en janvier, et les maintiendra alignés sur les patches d'avril. **Azul** a publié un billet sur Java 24 et propose son JDK Zulu 24 dès la sortie officielle (Six JDK 24 Features You Should Know About - Azul | Better Java Performance, Superior Java Support (https://www.azul.com/blog/six-jdk-24-features-you-should-know-about/#:%7E:text=Swiss%20watch,JEPs)), tout comme Eclipse Temurin. De plus, Azul a souligné dans un article six fonctionnalités de Java 24 particulièrement intéressantes pour les développeurs (couvrant AOT, Stream, Virtual Threads, Unsafe, 32-bit) (Six JDK 24 Features You Should Know About - Azul | Better Java Performance, Superior Java Support (https://www.azul.com/blog/six-jdk-24-features-you-should-know-about/#:%7E:text=JEP%20486%3A%20Permanently%20disable%20the,security%20manager)) (Six

JDK 24 Features You Should Know About - Azul | Better Java Performance, Superior Java Support (https://www.azul.com/blog/six-jdk-24-features-you-should-know-about/#:%7E:text=JEP%20491%3A%20Synchronize%20virtual%20threads,without%20pinning)), ce qui a alimenté des discussions sur le fait de savoir s'il est déjà utile de passer sur Java 24 ou s'il vaut mieux attendre Java 25 LTS.

En conclusion sur les événements, **mars 2025 a démontré la vigueur de l'écosystème Java** : les conférences reprennent en présentiel avec du contenu de haut niveau, Oracle et la communauté communiquent activement sur les nouveautés, et les éditeurs tiers (Red Hat, IBM, JetBrains, Azul, etc.) participent en diffusant leurs propres nouvelles. L'utilisateur Java a donc eu ce mois-ci de nombreuses occasions d'apprendre et d'échanger, que ce soit via JavaOne, via son JUG local ou en ligne.

Bonnes pratiques et conseils mis en avant par la communauté

Au-delà des annonces de nouvelles versions, la **communauté Java** a mis l'accent en mars 2025 sur plusieurs **meilleures pratiques, patterns émergents et conseils d'architecture**. Voici quelques thèmes récurrents dans les discussions, articles de blog et retours d'expérience du mois :

- **Adoption des threads virtuels (Loom) et patterns de concurrence** : Avec la généralisation de Loom dans Java 21+, de nombreux développeurs partagent désormais des retours d'usage de **Virtual Threads** en production. La communauté souligne quelques bonnes pratiques : utiliser des pools d'exécuteurs structurés (par exemple, un par grand type de tâche) au lieu d'un seul pool global, pour mieux isoler les charges ; tirer parti du couplage entre Virtual Threads et **StructuredTaskScope** (Concurrence Structurée) pour gérer proprement l'annulation et les timeouts de groupes de tâches liés (Java 24 Delivers New Experimental and Many Final Features - InfoQ (https://www.infoq.com/news/2025/03/java24-released/#:%7E:text=feature%20simplifies%20concurrent%20programming%20by,of%20the%20proposed%20API%20changes)) ; surveiller la consommation mémoire (un thread virtuel consomme moins qu'un thread OS, mais une mauvaise utilisation peut quand même mener à des milliers de threads inutiles si mal géré). Un conseil fréquent est de **tester progressivement** l'introduction de threads virtuels sur des flux de traitement isolés avant de l'étendre partout. Par exemple, commencer par gérer les appels I/O (HTTP, DB) dans des virtual threads, tout en conservant l'existant, puis étendre. De plus, attention est attirée sur certaines bibliothèques non thread-safe ou qui pourraient bloquer : la communauté maintient des listes de compatibilité (beaucoup de libs sont Loom-friendly désormais, mais quelques connecteurs JDBC ou drivers plus anciens peuvent encore poser problème). En bref, **Loom est prêt** et la meilleure pratique émergeante est de l'utiliser pour simplifier la concurrence (adopter un code impératif plutôt que réactif pur si le besoin de réactivité venait uniquement de limitations de threads). Toutefois, chacun souligne l'importance de profiler et surveiller pour éviter d'ouvrir trop de connexions ou d'allouer trop de threads malgré tout.

- **Concurrence structurée vs flux réactifs** : Un débat d'architecture est réapparu avec l'arrivée de la concurrence structurée en preview. Faut-il continuer à investir dans les frameworks réactifs (RxJava, Reactor) ou simplifier le code avec Loom et StructuredTaskScope ? La tendance n'est pas à l'opposition frontale : la **bonne pratique recommandée** est de choisir l'outil en fonction du cas d'usage. Pour des **services très I/O-bound**, Loom + `StructuredTaskScope` offrent un code plus lisible et des performances quasi équivalentes au réactif tout en évitant l'effet *"callback hell"*. En revanche, pour des **pipelines de streaming de données** ou des besoins de **haute concurrence fine-grain** (genre 100k événements/s), les frameworks réactifs conservent l'avantage de la back-pressure native et de l'écosystème d'opérateurs riche. Ainsi, la communauté conseille de ne pas systématiquement tout réécrire, mais d'**introduire les virtual threads dans de nouveaux développements** ou lorsque la simplicité est préférable à la performance micro-optimisée. Certains projets combinent d'ailleurs les deux : par exemple des microservices globalement impératifs avec Loom, mais qui utilisent des bibliothèques réactives pour un composant précis (messagerie, etc.). Ce pragmatisme reflète la maturité du discours : **le pattern émergent est d'écrire du code concurrent plus simple à comprendre**, et Java offre désormais plusieurs façons de le faire (structurée, réactive, ou même une combinaison via des adaptateurs).

- **Observabilité "built-in"** : Comme mentionné auparavant, toutes les stacks Java intègrent maintenant facilement des solutions d'observabilité. La meilleure pratique soulignée est d'**activer l'observabilité dès le début du projet** plutôt que de le rajouter après-coup. En pratique, cela signifie : configurer Micrometer et/ou OpenTelemetry dans vos microservices, utiliser les métriques d'**instruments standard** (JVM memory, CPU, threads, etc.) qui sont fournis out-of-the-box, et propager les contextes de trace dans vos appels (souvent géré automatiquement par les frameworks). La communauté a mis en avant les nouveaux outils comme l'**OpenTelemetry Java Agent** qui peut autoinstrumenter une application sans changer le code. Les retours insistent toutefois sur la **gestion du volume de données** : il est facile d'inonder son système de monitoring avec trop de métriques ou de traces, donc il faut définir quelles métriques clés sont nécessaires (SLA, SLO). On recommande d'appliquer les **principes RED** (Rate, Errors, Duration) pour les services et **USE** (Utilization, Saturation, Errors) pour la JVM, comme guide des choses à monitorer. Par ailleurs, la capacité à suivre une requête de bout en bout (tracing distribué) est de plus en plus considérée comme un must-have en microservices – d'où l'adoption large du W3C TraceContext. Bref, **instrumenter son application Java n'est plus optionnel** en 2025, c'est une pratique standard encouragée par tous les acteurs.

- **Architecture modulaire et monolithes modulaires** : Avec la complexité rencontrée par certains projets microservices, on observe un regain d'intérêt pour les **"modular monoliths"** – en clair, construire une application monolithique mais fortement modulairisée en composants indépendants. Spring Modulith (mentionné plus haut) est justement une réponse à cela, fournissant un cadre pour définir des modules explicitement, publier des événements entre modules, tester l'isolation, etc. Les experts conseillent que **même si vous partez en microservices, concevez initialement votre application comme un modulith** : cela permet de bien tracer les frontières de responsabilité. Par la suite, si un jour un module doit devenir un service séparé, la séparation sera bien plus facile. D'autre part, **Java Module System (JPMS)** n'est pas mort : certains projets l'adoptent pour cloisonner physiquement les modules, surtout en desktop ou sur de très grosses bases de code serveur. La meilleure pratique dans ce domaine est cependant de ne pas trop complexifier son build avec JPMS tant que ce n'est pas nécessaire – beaucoup choisissent des modules logiques (packages, conventions, modulith) sans activer `module-info.java` partout, afin de ne pas se heurter aux problèmes de modules transitoires et réflexion interdite. En résumé, l'idée d'une architecture modulaire a le vent en poupe, comme antidote à la dérive des microservices non maîtrisés. Les outils Spring Modulith, ArchUnit (pour vérifier l'architecture) ou encore l'approche DDD (contextes délimités) sont cités comme très utiles pour mettre cela en pratique.

- **Mise à jour et maintenance proactive** : Un autre conseil répété : **garder ses dépendances à jour**. Avec la cadence rapide des releases (tous les frameworks majeurs publient des versions mineures tous les un à trois mois), il peut sembler tentant de ne pas suivre. Pourtant, la communauté souligne l'importance des mises à jour de sécurité (Spring, Quarkus, etc. publient régulièrement des patches corrigeant des vulnérabilités). Des outils tels que Renovate, Dependabot, ou Maven Versions Plugin sont recommandés pour surveiller les nouvelles versions. Adopter les versions LTS des frameworks si disponibles (Quarkus 3.20 LTS par exemple) peut faciliter le support. Une bonne pratique DevOps associée est de **mettre en place des pipelines de tests robustes** de sorte qu'une montée de version mineure puisse être validée automatiquement. Ainsi on réduit la peur de mettre à jour. Dans l'ensemble, l'idée est que l'environnement Java bouge vite mais **en tirer parti demande de la proactivité** : les projets stagnants sur d'anciennes versions finissent par payer une dette plus lourde à la migration.

- **Nouveaux patterns de conception** : En mars, quelques discussions ont tourné autour de **patterns** liés aux nouveautés du langage. Par exemple, l'usage judicieux des `sealed classes` et `pattern matching` pour implémenter des états finis (state machines) de manière concise. Ou encore l'apparition du pattern "**TAO**" (**Thread-Local Active Object** pattern) avec les Scoped Values, qui propose une alternative aux ThreadLocal pour partager du contexte immuable entre threads dans un scope défini (Oracle Releases Java 24 (https://www.oracle.com/news/announcement/oracle-releases-java-24-2025-03-18/#:%7E:text=,in%20AI%20inference%20and%20compute)). Ce pattern, une fois que Scoped Values sera finalisé, pourrait remplacer bon nombre d'utilisations de ThreadLocal (jugé verbeux et propice aux fuites). On voit aussi des discussions sur l'organisation du code avec les **Records** – par exemple utiliser des records pour tous les *DTO immuables* afin de bénéficier de l'auto-génération des méthodes `equals/hashCode/toString`. La communauté encourage l'utilisation de ces nouvelles constructions du langage qui rendent le code plus clair et moins sujet aux erreurs, et considère qu'il s'agit là de la *modernisation* du style de programmation Java (tout comme l'utilisation systématique des `var` locaux lorsqu'évident, pour alléger la syntaxe).

- **Sécurité et vulnérabilités** : Enfin, du côté des bonnes pratiques en **sécurité**, on rappellera qu'en mars s'est tenu l'événement "Java Security Summit" (en virtuel) où il a beaucoup été question de la sécurisation de la chaîne logistique (supply chain). Les experts recommandent d'adopter des outils comme **Sigstore**/Cosign pour signer les artefacts, d'utiliser des scans de vulnérabilités (OWASP Dependency Check, Snyk) intégrés au build, et de bien suivre les bulletins de sécurité des JVM (les CPU d'Oracle). Par exemple, Java 24 a comblé certaines failles et intègre de nouvelles algos de chiffrement post-quantum – même si ce n'est pas un besoin immédiat, rester sur les dernières versions évite d'exposer des failles connues. La pratique d'avoir une **"SBOM" (Software Bill of Materials)** pour ses applications Java (via Maven ou Gradle plugin) se répand, car elle facilitera la réponse en cas de nouvelle faille de type Log4Shell. Autrement dit, la sécurité applicative Java se professionnalise et cela passe par des processus outillés.

En synthèse, la communauté Java en mars 2025 conseille de **profiter des avancées récentes** (Loom, records, etc.) pour écrire un code plus simple et maintenable, tout en gardant un œil sur les bonnes pratiques historiques (tests, modularité, monitoring, sécurité). Les patterns émergents ne remplacent pas forcément les anciens du jour au lendemain, mais offrent de nouvelles options pour résoudre des problèmes courants. Il est intéressant de noter que l'écosystème Java intègre progressivement des idées d'autres univers (par exemple la concurrence structurée vient du monde Go/CSP, la tendance modulith est une redécouverte de principes d'architecture logicielle plus que de la technique pure). Ainsi, la **meilleure pratique ultime** reste de *rester curieux et ouvert* aux évolutions, de tester dans de petits projets ou POC, afin d'évaluer ce qui apporte réellement de la valeur dans votre contexte.

Analyse critique : impacts réels vs nouveautés anecdotiques

Face à cette multitude de nouveautés de mars 2025, il convient de prendre du recul et d'**évaluer leur impact réel** sur les projets Java. Quelles sont les avancées qui vont concrètement améliorer la vie des développeurs ou la qualité des applications, et lesquelles sont plus anecdotiques ou à effet à long terme ?

Nouveautés les plus impactantes à court terme :

- *Virtual Threads (Loom) et améliorations de la concurrence* – Bien que Loom ait été introduit en Java 19/20 puis finalisé en Java 21, c'est vraiment en 2024-2025 que son impact se fait sentir. Désormais, la plupart des frameworks Web supportent les threads virtuels (Tomcat, Jetty, Netty, etc. ont levé les limitations ou fourni des intégrations spécifiques). Le fait que Spring, Helidon, Quarkus promeuvent leur usage signifie qu'on est à un point d'inflexion. Pour un projet Java backend, la possibilité de gérer un grand nombre de requêtes concurrentes de façon simple (un thread par requête, sans se soucier du pool) est un **changement de paradigme** qui simplifie potentiellement beaucoup de code et réduit la nécessité d'apprendre des librairies réactives complexes si on n'en a pas envie. C'est sans doute l'une des avancées les plus tangibles pour les développeurs ordinaires : on peut écrire du code concurrent plus facilement, et ça **fonctionne en production**. On peut donc s'attendre à ce que dans les prochains mois, de plus en plus de projets migrent vers Java 21+ pour tirer parti de Loom. À cet égard, les améliorations apportées dans Java 24 (synchronized sans pinning, etc.) renforcent encore la viabilité de Loom pour tous les cas d'utilisation (y compris ceux qui utilisent des moniteurs synchronisés). On peut estimer que **Loom aura un impact aussi profond que l'introduction des threads eux-mêmes en Java dans les années 90**, car il modifie certains choix d'architecture (moins de besoin d'asynchrone, ou exploitation de centaines de threads là où 50 suffisaient mais sans conséquence négative).

- *Sortie de Java 24 et preview de fonctionnalités à venir* – Java 24 en lui-même, étant non-LTS, ne sera pas massivement déployé en production. Cependant, il donne un aperçu concret de ce qui attend les développeurs dans Java 25 LTS. Des fonctionnalités comme les **patterns sur types primitifs** ou la **concurrence structurée** vont simplifier davantage la syntaxe et la sûreté du code. Leur impact sera réel une fois finalisés (probablement Java 25 ou 26). De même, les efforts comme **Stream gatherers** ou **Classfile API** sont précieux pour des niches de développeurs (ceux qui écrivent des libs de manipulation de bytecode, ou qui manquaient d'une opération custom en Stream). L'impact est donc inégal selon les features : pour la plupart des développeurs d'applications métier, on peut considérer que **Java 24 n'apporte pas de révolution immédiate** (ils ne pourront utiliser ses preview qu'en activant un flag, ce que peu feront en prod). En revanche, les améliorations de performance de la JVM (GC, AOT loading) bénéficient automatiquement à qui passera sur JDK 24. Mais étant non-LTS, peu de monde fera ce saut, préférant attendre Java 25. En somme, Java 24 est important pour l'écosystème (dernière étape avant la prochaine LTS), mais **son impact est surtout à moyen terme**, lorsque ses features preview deviendront permanentes. A court terme, c'est un terrain de jeu pour expérimenter – ce que peu d'entreprises en production feront, mais que les développeurs peuvent tester localement.

- *Spring Boot et Spring Framework en transition* – Spring reste le framework le plus utilisé en entreprise. Les versions 3.4/3.5 de Boot et 6.2 de Framework en mars apportent des améliorations incrémentales, mais rien de bouleversant. Ce qui est plus impactant, c'est la perspective de **Spring 7 / Boot 4 fin 2025**. Cela signale aux équipes qu'une migration majeure sera à prévoir en 2026 pour rester à jour (passage Jakarta EE 11, potentiellement JDK 17 mini, etc.). Pour l'instant, les changements Spring du mois ont un impact faible : ce sont essentiellement des corrections et un alignement sur Java 21. On peut voir que Spring investit sur l'observabilité, la sécurité (JWT), mais ce sont des continuations logiques. La réelle nouveauté serait Spring AI si cela devient un projet concret, mais en l'état ce n'est qu'un milestone, pas encore de quoi changer la donne dans les projets. **Conclusion** : Spring en mars 2025 c'est du *business as usual*, stable et fiable, sans rupture. Les projets Spring actuels continueront à tourner de la même façon ; peut-être profiteront-ils de Boot 3.4.4 pour éviter un warning JDK24, mais c'est tout. L'impact majeur viendra plus tard avec Spring 7.

- *Jakarta EE 11* – Le fait que Jakarta EE 11 arrive est important pour les éditeurs de serveurs et les entreprises encore sur Java EE 8 ou Jakarta EE 9/10. **À court terme, l'impact est limité** car il faudra du temps pour que les serveurs Jakarta EE 11 soient disponibles et stables (sans parler de la certification des produits commerciaux type WebSphere, JBoss EAP, qui arrivera peut-être en 2026). Néanmoins, Jakarta EE 11 en soi est crucial pour garder l'écosystème Java entreprise à jour : il apporte la compatibilité Java 17/21, ce qui était très attendu. Son impact réel sera de donner un chemin de migration aux grosses applis legacy vers du moderne (on pourra amener un vieux WAR Java EE sur un serveur Jakarta EE 11 tournant sur JDK 17). Mais ce mouvement sera lent ; beaucoup d'organisations attendent généralement un an ou deux avant d'adopter la nouvelle version EE, le temps que les outils autour suivent (ORM, serveurs stables, etc.). Ainsi, en mars 2025, c'est surtout **un signal positif et rassurant**

(Java EE évolue enfin, ne stagne pas). Les développeurs ont une meilleure visibilité sur l'avenir de Jakarta (EE 12 planifié, etc.), ce qui peut influencer leurs choix (rester sur Jakarta vs basculer sur Spring). Pour ceux qui ont déjà adopté Quarkus/Spring Boot au lieu d'app serveurs, Jakarta EE 11 aura peu d'impact direct – sauf via les libs partagées (par ex, JSF 4, JPA 3.1 peuvent être utilisés hors serveurs aussi).

- *Quarkus 3.20 LTS* – Pour la communauté Quarkus, c'est une étape importante. Avoir une version LTS stable signifie que Red Hat et les contributeurs vont prioriser le support de celle-ci. Les améliorations apportées (config mapping unifié, Observabilité OTel, etc.) améliorent la **qualité de vie des développeurs Quarkus** et la cohérence du framework. L'impact en production est que Quarkus 3.20 est sans doute la version recommandée pour les nouveaux projets ou les migrations depuis Quarkus 2. Les nouveautés ne changent pas radicalement la façon de coder (c'est surtout interne), mais éliminent des irritants (ex: plus de config legacy, mocking plus puissant en tests, base images plus sécurisées). On peut donc dire que **Quarkus 3.20 consolide l'avance de Quarkus sur le terrain du cloud-native Java** : ceux qui l'utilisent verront un bénéfice en maintenance et support, et ceux qui hésitaient à l'adopter pourraient être convaincus par sa stabilité LTS. Par contre, rien de totalement inédit, pas de nouvelle fonctionnalité utilisateur spectaculaire – c'est avant tout de la **maturation**. Donc impact fort en fiabilisation, faible en disruption.

- *Micronaut et autres frameworks* – Micronaut 4.8 apporte de bonnes choses (debug DI, etc.) mais cela reste assez technique. L'impact est surtout pour les développeurs Micronaut qui pourront plus facilement comprendre la magie du framework. Micronaut continue d'être un choix solide mais reste moins utilisé que Spring ou Quarkus ; son impact global sur l'écosystème est donc modéré. Néanmoins, ses idées (AOT, pas de réflexion) influencent Spring Native et d'autres, donc indirectement Micronaut pousse l'écosystème à être plus performant. On peut dire que **Micronaut prépare tranquillement l'avenir** (avec SourceGen) sans révolutionner le présent.

- *Outils et bibliothèques* – Parmi la liste, il faut avouer que certaines nouveautés sont **anecdotiques pour la majorité des développeurs**. Par exemple, FluentJdbc est sympa, mais beaucoup de devs vont continuer à utiliser JPA/Hibernate ou Spring JDBC. Son impact reste limité à ceux qui cherchent précisément ce genre d'outil, ce qui est une minorité. Idem pour LangChain4j – intéressant pour introduire l'IA en Java, mais aujourd'hui ça reste un usage de niche (tous les projets ne vont pas intégrer du LLM). Par contre, l'effort autour de l'observabilité (Micrometer/Otel) est très impactant car quasi tout le monde en a besoin : ce n'est plus une niche, c'est devenu mainstream dans le dev d'apps distribuées. Ainsi, la disponibilité de connecteurs OTel dans Quarkus ou Spring va faciliter la vie de milliers de développeurs, même si c'est discret. D'autres outillages comme JUnit qui se met à jour ou Gradle qui améliore le config-cache, ce sont des impacts cumulatifs sur la productivité (builds plus rapides, tests plus robustes) – *l'effet n'est pas spectaculaire mais bien réel* sur le long terme.

- *Côté négatif / nuancé* : Y a-t-il des choses présentées comme des nouveautés qui sont en fait peu utiles ? On peut se poser la question pour certains JEPs de Java 24 par exemple : la suppression du Security Manager, certes importante pour nettoyer l'API, n'impacte pratiquement personne (il y avait bien longtemps que plus grand monde ne l'utilisait pour sécuriser une app). De même, le **retrait du support 32-bit** – c'est mentionné, mais en 2025 très peu de déploiements Java tournent encore en 32 bits. Donc l'impact sera quasi nul, si ce n'est clarifier la maintenance de la JVM. Les algos post-quantiques : c'est prudent de les intégrer tôt, mais soyons honnêtes, **personne ne va utiliser ça en prod en 2025** car ce n'est pas standardisé complètement et les perfs sont pas optimales. C'est un pari sur le futur, impacté possiblement dans 10 ans. Donc pour l'instant c'est anecdotique pour le dev quotidien. D'autres choses comme *Module Import* en preview : c'est un sucre syntaxique pour un cas (les projets non-modularisés voulant importer les modules). Utile en salle de cours, mais pas un game-changer industriel. On pourrait aussi parler de **Spring AI** – c'est intriguant, mais on ne sait pas si ça aura de l'adoption ou si ça va vivoter comme Spring Flo ou d'autres projets annexes. Donc impact très incertain à ce stade.

En synthèse, les **nouveautés réellement impactantes en mars 2025** sont celles qui **vont dans le sens d'une simplification et performance accrues** : Loom (virtual threads), l'adoption plus simple de Java 17/21 via Jakarta EE 11, la convergence des outils d'observabilité, l'émergence de solutions pour modulariser autrement (modulith). Celles-ci ont déjà ou auront dans l'année un effet concret sur comment on développe en Java (plus de parallélisme simple, plus de monitoring intégré, moins de complexité configurationnelle).

Les **nouveautés plus anecdotiques** ou à impact différé sont : certains détails du JDK (crypto PQC, Applet API out), de nouvelles libs sympas mais de niche (LangChain4j, FluentJdbc), ou encore des annonces de versions futures (Spring 7, Jakarta EE 12) dont l'effet se fera sentir plus tard.

Enfin, il convient de relativiser : ce qui est *anecdotique pour l'ensemble de la communauté* peut être *crucial pour un domaine particulier*. Par exemple, **TornadoVM 1.1.0** (cité dans une news) permet d'exécuter du bytecode Java sur GPU – c'est un sujet de pointe, qui concerne peu d'applications aujourd'hui. Mais pour ceux qui font du calcul intensif, c'est potentiellement révolutionnaire. On ne l'a pas détaillé ici car c'est très spécialisé. De même, la sortie du **JDK 25 early-access build 18** signalée par InfoQ (Java News Roundup: WildFly 36, Spring Milestones, Betas for Open Liberty, Hibernate, LangChain4j - InfoQ (https://www.infoq.com/news/2025/04/java-news-roundup-apr07-2025/#:%7E:text=This%20week%27s%20Java%20roundup%20for,0)) montre que Java 25 se prépare activement, mais en pratique ça n'affecte encore personne en dehors des contributeurs OpenJDK.

Donc, notre analyse est que la plupart des nouveautés de mars 2025 vont dans le bon sens (**amélioration incrémentale de l'existant** plutôt que rupture), ce qui est une bonne chose pour les projets : on peut adopter ces changements de façon graduelle sans casse. Les quelques éléments vraiment nouveaux (e.g. adoption de threads virtuels) sont suffisamment mûrs pour être utilisés prudemment. Il n'y a pas de « hype vide » ce mois-ci – chaque annonce correspond à un besoin réel, même si l'ampleur varie du grand public au spécifique.

Impact sur les roadmaps de projet : Un chef de projet ou tech lead Java en mars 2025 devra surtout intégrer dans sa roadmap :

- La planification de la montée de version Java (passage à Java 21 LTS, puis Java 25 LTS en 2025/2026).
- La préparation de migrations de frameworks majeurs (ex : Spring Boot 4 l'année prochaine, Jakarta EE 11 si concerné).
- L'intégration de l'observabilité et d'une architecture plus simple (peut-être refondre des microservices trop complexes en moduliths).
- Former l'équipe aux nouveaux outils (Loom, etc.) pour en tirer parti.
- Surveiller l'opportunité d'introduire des technos émergentes (si IA potentiellement utile, garder un œil sur Spring AI ou LangChain4j, etc.). En éliminant le bruit, c'est là que se situent les impacts tangibles.

Roadmaps officielles et perspectives à venir

Plusieurs éditeurs et projets ont publié ou confirmé leurs **roadmaps officielles** récemment, offrant de la visibilité sur les évolutions à venir dans l'écosystème Java. Récapitulons les points saillants des feuilles de route pour les prochains mois/années :

- **Java (OpenJDK)** : Le cycle de Java étant désormais bien cadencé, on sait que la prochaine version sera **Java 25 en septembre 2025** et ce sera une **LTS**. Aucune incertitude là-dessus, la JCP l'a planifié (Java 21 LTS -> Java 25 LTS, tous les 2 ans). Java 25 devrait intégrer ce qui est en preview dans 22-24, notamment probablement rendre final la Concurrence Structurée, les Pattern Matching étendus, les API de cryptographie post-quantique, etc., et peut-être d'autres JEP en cours (String Templates ? Value Objects ?). Bien sûr, on aura Java 26 en mars 2026, etc. Pour suivre ces évolutions, le site OpenJDK et la JDK 25 project page sont à consulter. Oracle maintient également le support de **Java 17 jusqu'en 2029** et de **Java 21 jusqu'en 2031** (pour l'offre payante Oracle Java SE Subscription), ce qui donne une perspective longue aux entreprises sur ces LTS. Enfin, Oracle a évoqué la vision du **Projet Leyden** (améliorer le temps de démarrage et la taille des applications) dont on voit les premiers pas (AOT loading en JDK 24). On peut s'attendre à plus d'outils et d'options de compilation native ou d'images statiques d'ici Java 26/27, même si rien n'est officiel pour l'instant.

- **Spring** : Comme détaillé précédemment, la roadmap Spring est clairement annoncée par son lead Juergen Hoeller : **Spring Framework 7.0 GA en novembre 2025** (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=At%20the%20same%20time%2C%20we,0)) (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=We%20will%20upgrade%20our%20baseline,the%20recently%20released%20JSpecify%20annotations)). Cela impliquera un Spring basculant sur Jakarta EE 11 et supportant le JDK 25 (tout en gardant compatibilité JDK 17). Spring 7 se concentrera sur la continuité (pas de réécriture complète du framework), avec nettoyage de vieilles APIs, amélioration du support de GraalVM native, possiblement de nouveaux modules (Spring AI ?). **Spring Boot 4.0** sortira juste après Spring 7 (fin 2025) (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=A%20first%20Spring%20Framework%207,0%20GA)), ce qui signifie probablement un Spring Boot nécessitant Java 17+ et apportant peut-être un packaging différent (à voir si la structure des starters évolue). En attendant, **Spring Boot 3.5** est prévu pour mai 2025 (basé Spring 6.2) et ce sera la dernière release mineure de la v3. Ensuite, **Spring Boot 3.6** n'est pas prévu ; on passera sur Boot 4. Spring Cloud a également calé son calendrier : **Spring Cloud 2025.0 (Northfields)** sortira vers mai 2025 (avec Boot 3.5), et **Spring Cloud 2025.1** en fin 2025 avec Boot 4/Spring 7 (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=A%20first%20Spring%20Framework%207,0%20GA)). Donc les utilisateurs Spring peuvent planifier : 2024-2025 sur Spring 6/Boot 3, puis migration en 2026 sur Spring 7/Boot 4. Spring Security, Data, Batch et consorts suivront ce train. Les roadmaps de Spring Data par exemple évoquent déjà une version **2025.1.0** en fin 2025 qui aura Jakarta EE 11 en minimum (Spring Data 2025.1.0-M1 released (https://spring.io/blog/2025/01/24/spring-data-2025#:%7E:text=Spring%20Data%202025.1.0,requirements%20to%20Jakarta%20EE%2011)). On voit aussi émerger de nouveaux projets Spring (Spring AI, Spring CLI, etc.) – ils seront intégrés si le marché les adopte. Spring continue de mettre l'accent sur le **support multi-runtime** (JVM et native), donc attendez-vous à ce que Spring 7 facilite encore la compilation native et consomme moins de ressources (leur collaboration avec GraalVM et le projet Leyden va dans ce sens (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=applications%2C%20we%20intend%20to%20base,the%20recently%20released%20JSpecify%20annotations))).

- **Jakarta EE** : La **roadmap Jakarta EE 11** est en phase de conclusion (release en mai 2025). La question se pose : quelle suite et quand ? L'équipe Jakarta a déjà lancé le planning de **Jakarta EE 12** en ce début 2025 (Jakarta EE | agilejava.eu (https://www.agilejava.eu/category/jakarta-ee/#:%7E:text=Image)). S'il y a une volonté d'accélérer, Jakarta EE 12 pourrait arriver dès fin 2026 ou 2027, mais cela dépendra du contenu. Ivar Grimstad (chef de projet Jakarta EE) indique qu'après le TCK de EE 11, on va se concentrer sur définir EE 12 (Jakarta EE | agilejava.eu (https://www.agilejava.eu/category/jakarta-ee/#:%7E:text=Plan%20reviews%20for%20Jakarta%20EE,15%20deadline%20is%20fast%20approaching)). On peut supposer que Jakarta EE 12 prendra en charge Java 21 ou 25 comme minimum (peut-être Java 21 minimum, pour profiter de l'API Foreign, etc., ou Java 25 si c'est raisonnable). Des thèmes probables : une spécification de configuration unifiée (pour remplacer MicroProfile Config ?), améliorer Jakarta NoSQL, supporter gRPC ou GraphQL, et retirer d'anciennes choses (peut-être le vieillissant Jakarta Activation ou les EJB entity beans qui sont déjà obsolètes). A plus court terme, la **roadmap des implémentations** Jakarta EE 11 est : GlassFish 8.0 final ~mi-2025, Open Liberty 23.x avec feature complete Jakarta EE 11 sans doute l'été 2025, WildFly 37 stable vers fin 2025, etc. Donc les utilisateurs de serveurs devraient tester Jakarta EE 11 fin 2025 / début 2026 dans leurs environnements.

- **Quarkus** : Red Hat a indiqué un **rythme d'une LTS par an** environ pour Quarkus. Après la 3.20 en mars 2025, on peut s'attendre à ce que la prochaine LTS soit **Quarkus 4.0 en 2026** (peut-être mars 2026). D'ailleurs, Quarkus 4 pourrait coïncider avec le passage à Jakarta EE 11 (actuellement Quarkus 3 utilise encore Jakarta EE 10 pour certaines dépendances, une 3.x future pourrait supporter Jakarta EE 11 en partial, mais Quarkus 4 c'est l'occasion de tout passer). Pour l'instant, la feuille de route Quarkus 3.x continue avec des versions mensuelles (3.21, 3.22…). On voit dans le planning GitHub qu'ils anticipent déjà jusqu'à 3.22 (Migration Guides · quarkusio/quarkus Wiki - GitHub (https://github.com/quarkusio/quarkus/wiki/Migration-Guides#:%7E:text=Migration%20Guides%20%C2%B7%20quarkusio%2Fquarkus%20Wiki,released%20on%20Jan%2029th%202025)). Red Hat va certainement aligner Quarkus sur RHEL et son écosystème – Quarkus 3.x accompagne Java 21 LTS, Quarkus 4.x accompagnera Java 25 LTS. Sur le plan des features, Quarkus 4 pourrait tenter d'intégrer encore plus **Loom** (par ex. permettre de choisir d'exécuter les requêtes HTTP sur virtual threads plutôt que son modèle actuel basé sur Vert.x event loop – un choix à définir). La roadmap n'est pas formellement publiée, mais les discussions communautaires mentionnent l'idée de simplifier le modèle de programmation (peut-être rendre toutes les routes REST non bloquantes sans utiliser Uni/CompletionStage). En tout cas, pour les utilisateurs Quarkus, la vision est claire : rester sur la 3.20 LTS pour la stabilité, et suivre les mises à jour mineures ; évaluer Quarkus 4 lorsqu'il sortira, avec l'idée d'une migration fin 2026.

- **Micronaut** : Le site officiel ne donne pas de dates, mais on peut extrapoler. Micronaut 4 est sorti en 2023, la **version 5.0** pourrait arriver en 2025 si on suit leur historique (Micronaut 1 en 2018, v2 en 2020, v3 en 2021, v4 en 2023). Micronaut 5 sans doute prendra Java 21 comme minimum et Jakarta EE 11+ (car Micronaut inclut des annotations Jakarta pour JAX-RS, etc.). L'accent sera sûrement mis sur le **GraalVM** (Micronaut est souvent le premier à supporter les nouvelles versions), et sur l'amélioration de la performance de compilation (d'où le projet SourceGen qui devrait peut-être être généralisé en v5). Une direction mentionnée est l'**interopérabilité avec Spring** (Micronaut propose un mode de fonctionnement dans Spring via le projet Oracle Coherence par ex.), cela pourrait être renforcé pour permettre à des apps Spring d'utiliser des features Micronaut et vice versa. A court terme, **Micronaut 4.x** va continuer avec des 4.8, 4.9… Il est déjà compatible Jakarta EE 10 ; la fondation Micronaut participera sûrement à MicroProfile et Jakarta EE pour s'assurer que leur produit fonctionne bien sur ces standards.

- **Autres** :

    - *Build tools* : Gradle a une roadmap ouvertement partagée. **Gradle 8.x** sera la série stable en 2025, avec un probable Gradle 9 fin 2025 apportant des changements plus importants (peut-être une unification avec Kotlin multiplateforme, ou des optimisations massives de configuration cache). Maven évolue plus lentement, la version 4.0 (majeure) est en alpha depuis longtemps – possiblement une RC en 2025.
    - *MicroProfile* : La version 6.1 de MicroProfile est sortie en 2023 avec Jakarta EE 10. On peut imaginer une MicroProfile 7 en 2025 alignée sur Jakarta EE 11. MicroProfile Config, Metrics, Fault Tolerance etc. vont poursuivre, mais leur influence baisse un peu car Jakarta EE reprend certaines de ces idées en son sein.
    - *Kotlin* : Ce n'est pas Java, mais très lié. Kotlin 2.0 pourrait sortir fin 2025 (JetBrains en parle en work in progress). Cela importera les *context receivers* etc. Cela pourrait influencer la communauté Java (par ex, si Kotlin devient plus performant ou ajoute des features que Java n'a pas, cela peut mettre une légère pression).

- *Cloud* : Les offres PaaS Java (Azure Managed Java, Google Cloud Run for Java, etc.) vont toutes intégrer Java 21 LTS durant 2025 si ce n'est déjà fait. L'adoption du JDK 17 sur Lambda AWS a été lente (dispo qu'en 2023), espérons que JDK 21 sur Lambda arrive plus vite. Cela peut jouer sur les roadmaps de déploiement des projets qui utilisent ces clouds.

En résumé, les **roadmaps officielles** indiquent une convergence en 2025-2026 vers :

- L'adoption généralisée de **Jakarta EE 11** et **Java 21/25** dans tous les frameworks.
- Une nouvelle génération de frameworks Spring, Quarkus, Micronaut alignés sur ces versions.
- Un accent sur la **performance** (démarrage, GraalVM, Leyden) et la **productivité développeur** (simplification du code, API plus haut niveau).
- Le début de l'intégration de **l'IA** dans l'environnement Java (outils spécialisés, libs de ML).
- Une attention continue sur la **sécurité** (Supply chain, etc.) dans les outils.

Les développeurs et architectes peuvent s'appuyer sur ces roadmaps pour planifier les évolutions de leur stack : par exemple, programmer une migration de Spring Boot 2 vers 3 sans tarder, car Boot 4 arrive bientôt ; s'assurer que leurs conteneurs et pipelines supportent Java 17 puis 21 ; évaluer Quarkus LTS si besoin d'une base stable ; et suivre de près les sorties LTS du JDK pour caler leur calendrier interne.

Récapitulatif final et impact sur la roadmap d'un projet Java

Pour conclure cette veille de mars 2025, remettons en perspective les éléments clés à retenir et comment ils peuvent influencer la **roadmap d'un projet Java** typique.

1. Plateforme Java (JDK) – La sortie de Java 24 confirme la cadence rapide d'innovation. Si vous maintenez un projet Java, **à court terme**, c'est Java 21 LTS qu'il faut cibler (ou Java 17 LTS a minima). Java 24 n'est pas LTS, mais il préfigure Java 25 LTS qui arrivera en septembre. **Sur la roadmap** : prévoyez dès maintenant de tester votre application sur Java 21 si ce n'est fait, et anticipez Java 25 (par ex. en réservant du temps en Q4 2025 pour migration). Les nouveautés comme Loom peuvent justifier des refontes de modules concurrents pour plus d'efficacité – identifiez dans votre application les endroits où un code complexe (basé sur CompletableFuture ou Reactive Streams) pourrait être simplifié via des threads virtuels ou structured concurrency, et planifiez des proof-of-concept en 2025. De même, si votre application subit des soucis de performance au démarrage, suivez les avancées du projet Leyden (AOT, etc.) qui pourraient, d'ici 1 à 2 ans, offrir des solutions natives ou pré-compilées dans le JDK lui-même.

2. Frameworks et runtimes – **Spring** reste largement dominant. La feuille de route dit : Spring Framework 7 / Boot 4 fin 2025. Donc, **si vous êtes sur Spring Boot 2 (Spring 5)**3. Plan de migration Spring/Java EE** – Si votre projet utilise **Spring Boot 2.x / Spring 5** ou **Java EE 8/Jakarta EE 9**, il est temps de planifier une mise à niveau. Spring Boot 2 n'est plus maintenu (fin de support en 2024), et Spring Boot 3 (Spring 6) offre la compatibilité Java 17+ et Jakarta EE 10. **Sur la roadmap 2025**, prévoyez la migration vers Spring Boot 3.1+ dès que possible. Cela vous positionnera idéalement pour Spring Boot 4 fin 2025, qui apportera Spring 7 et Jakarta EE 11 (From Spring Framework 6.2 to 7.0 (https://spring.io/blog/2024/10/01/from-spring-framework-6-2-to-7-0#:%7E:text=A%20first%20Spring%20Framework%207,0%20GA)). De même, si vous êtes sur un serveur Java EE ancien (WildFly 20, WebLogic 12c…), examinez les options Jakarta EE 10/11 : par exemple migrer vers **WildFly 27+ (Jakarta EE 10)** dans un premier temps, en attendant que les serveurs Jakarta EE 11 matures soient disponibles en 2025-2026. En clair, **mettez à jour vos frameworks fondamentaux** afin de bénéficier du support des nouvelles versions Java et des correctifs de sécurité.

4. Adoption des nouvelles fonctionnalités – Intégrez progressivement les nouveautés techniques qui apportent de la valeur. Par exemple, planifiez un **prototype utilisant les threads virtuels Loom** sur une partie non critique de l'application (un batch, un service annexe) pour évaluer le gain en simplicité et en performances. Si concluant, ajoutez dans la roadmap l'extension de Loom à d'autres composants lors d'un prochain sprint. De même, si vous n'avez pas encore d'**observabilité centralisée**, priorisez son implémentation : déployez un outil comme Prometheus/Grafana ou Elastic, et instrumentez l'app avec Micrometer/OpenTelemetry (souvent il suffit d'activer les starters Spring Boot ou l'extension Quarkus adéquate). Cela améliorera la supervision et permettra des diagnostics rapides en production. En ce qui concerne l'**architecture**, profitez des tendances modulaires : par exemple, utilisez **Spring Modulith** ou **ArchUnit** pour imposer des règles d'architecture dès maintenant, évitant l'endettement structurel. Si votre application microservices est trop morcelée, envisagez d'en **regrouper certaines en un modulith** plus cohérent – ce genre de refonte peut figurer en objectif sur 2 ou 3 releases de votre produit. Enfin, restez ouverts aux **nouvelles bibliothèques** pertinentes : sans tout refondre, vous pouvez introduire progressivement un outil comme FluentJdbc pour un nouveau module de persistence léger, ou tester LangChain4j sur un petit POC lié à du chatbot/IA si votre domaine s'y prête.

5. Veille technologique continue – Les évolutions Java récentes montrent l'importance de la veille. Assurez-vous que votre feuille de route intègre du **temps d'apprentissage** et d'expérimentation pour l'équipe. Cela peut être via des formations (ex : un workshop interne sur les nouveautés Java 21+), la participation à des conférences/JUG, ou des *sprints* d'innovation. Par exemple, bloquez un créneau après chaque sortie semestrielle du JDK pour que l'équipe examine les JEPs et discute de l'impact potentiel. Incitez vos développeurs à suivre les blogs de référence (InfoQ Java, Inside Java, blogs des Java Champions) pour capter les retours d'expérience de la communauté. Cette veille active permettra d'**adapter votre roadmap en continu** – par exemple, décider plus finement quand passer à Java 25 LTS une fois qu'il sera sorti et éprouvé par d'autres.

En synthèse, **mars 2025 confirme que Java a un écosystème en plein renouveau**, porté par l'arrivée de fonctionnalités de langage attendues depuis longtemps et par la modernisation des frameworks. Pour un projet Java, l'enjeu est d'**accompagner ce mouvement sans heurts** : mettre à niveau sa stack logicielle (JDK, frameworks) pour rester supporté et sécurisé, et tirer parti des innovations qui simplifient le développement (Loom, observabilité intégrée, patterns plus expressifs). Une roadmap bien pensée saura équilibrer **stabilité** et **innovation** : consolider les bases (LTS, versions stables) tout en introduisant progressivement les améliorations qui feront la différence en termes de **maintenabilité, de performance et de capacité à évoluer**. En adoptant cette approche, votre projet Java restera pérenne, performant et aligné avec les meilleures pratiques actuelles, prêt à relever les défis des prochaines années.