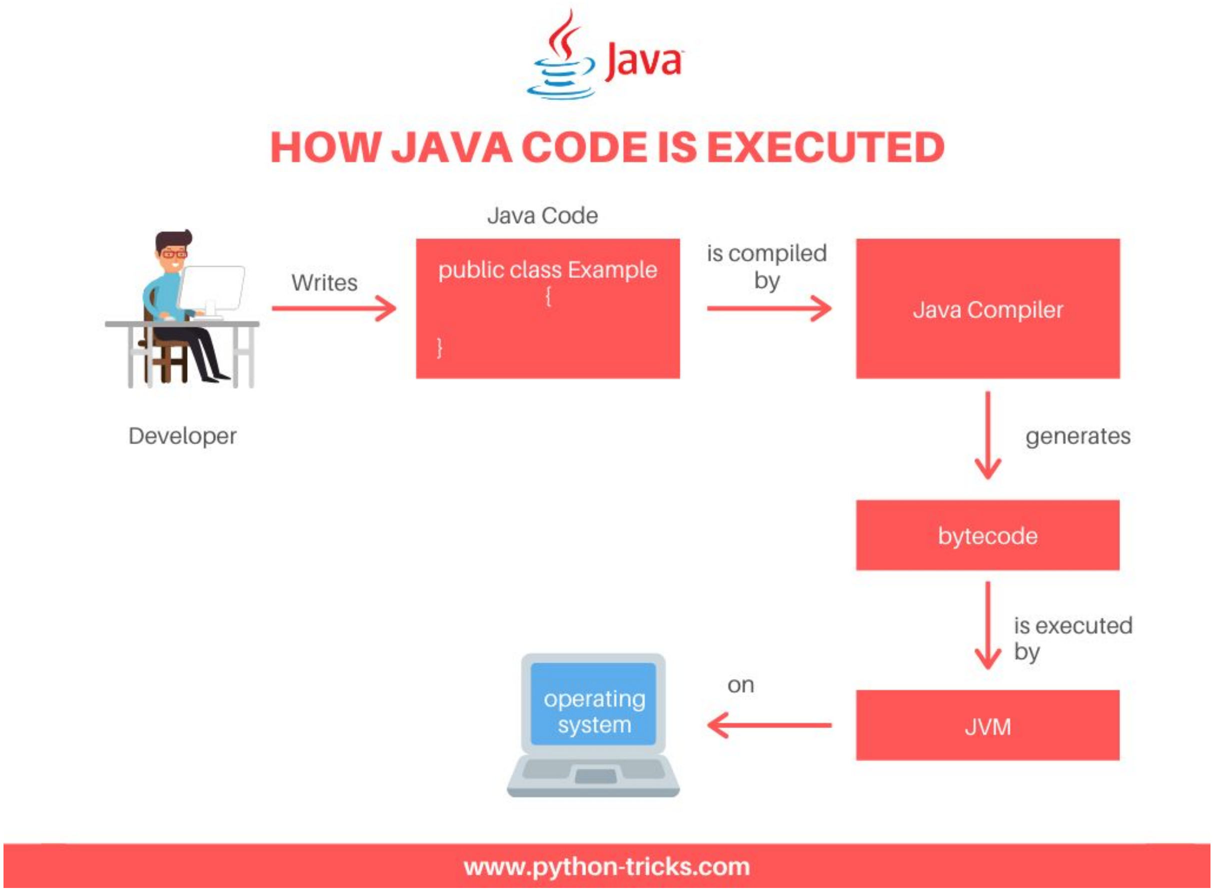
Migration from Java 8 to Java 17/21 Challenges

Introduction

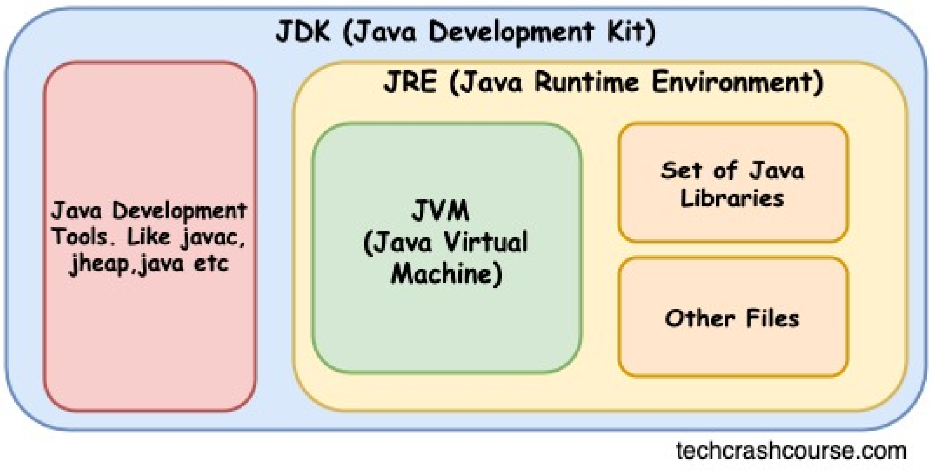
Software development is an ever-evolving field due to rapid technological advancements. One aspect of this evolution is the continuous updating of programming languages, such as transitioning from Java 8 to Java 17/21. While this migration comes with numerous potential benefits, it also presents unique challenges.

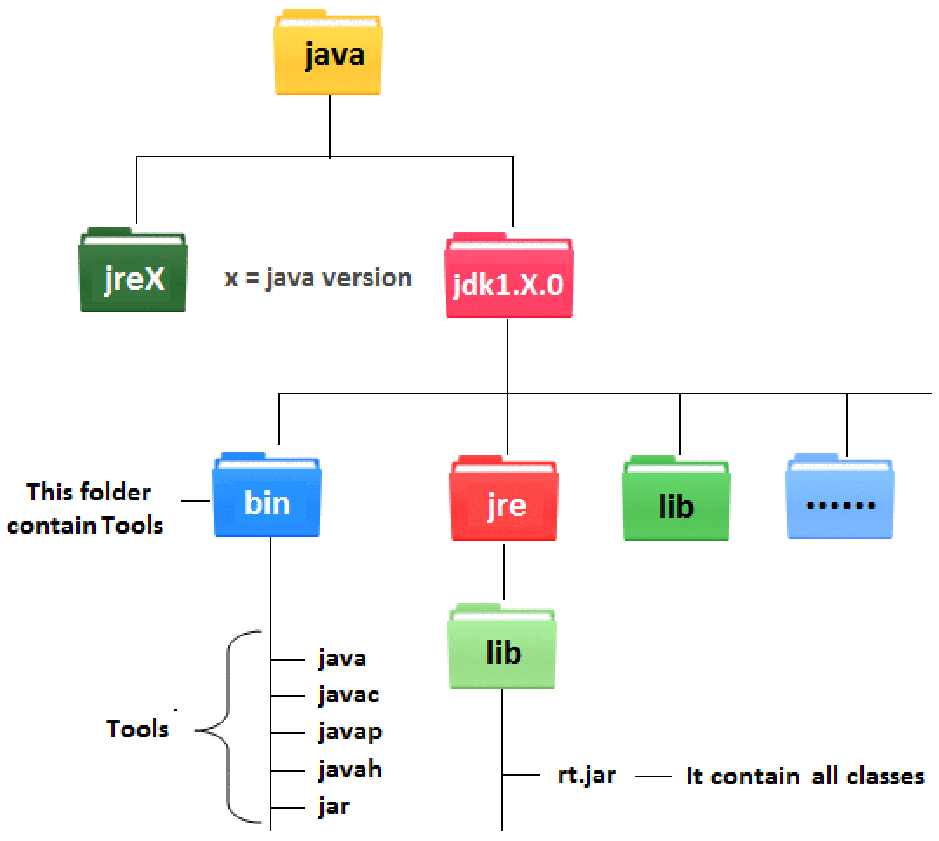


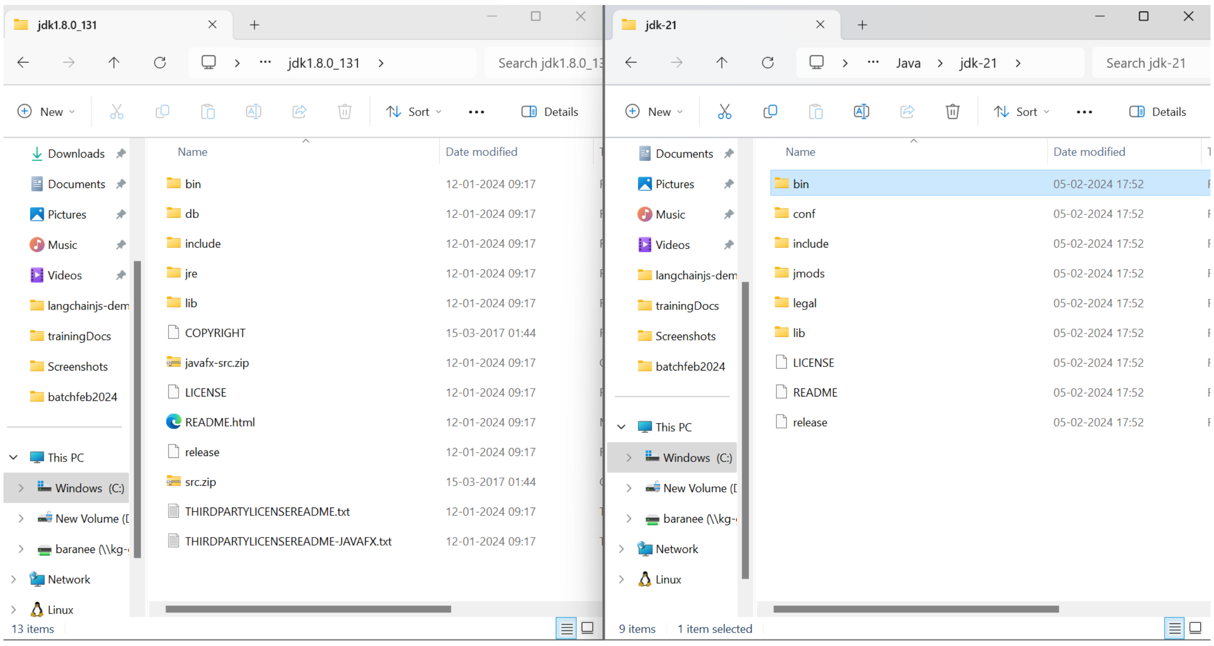
Overview of Java Versions

- **Java SE** is the foundation for all Java development.
- **Jakarta EE** extends Java SE for enterprise-level applications.
- **Java ME** targets mobile devices and embedded systems.

JDK JRE JVM


JDK (Java Development Kit)
Java Development Tools. Like javac, jheap,java etc
JRE (Java Runtime Environment)
JVM (Java Virtual Machine)
Set of Java Libraries
Other Files
techcrashcourse.com


java
jreX
x = java version
jdk1.X.0
This folder contain Tools
bin
jre
lib
......
lib
Tools
java
javac
javap
javah
jar
rt.jar
It contain all classes



jdk1.8.0_131

Name	Date modified
bin	12-01-2024 09:17
db	12-01-2024 09:17
include	12-01-2024 09:17
jre	12-01-2024 09:17
lib	12-01-2024 09:17
COPYRIGHT	15-03-2017 01:44
javafx-src.zip	12-01-2024 09:17
LICENSE	12-01-2024 09:17
README.html	12-01-2024 09:17
release	12-01-2024 09:17
src.zip	15-03-2017 01:44
THIRDPARTYLICENSEREADME.txt	12-01-2024 09:17
THIRDPARTYLICENSEREADME-JAVAFX.txt	12-01-2024 09:17

13 items

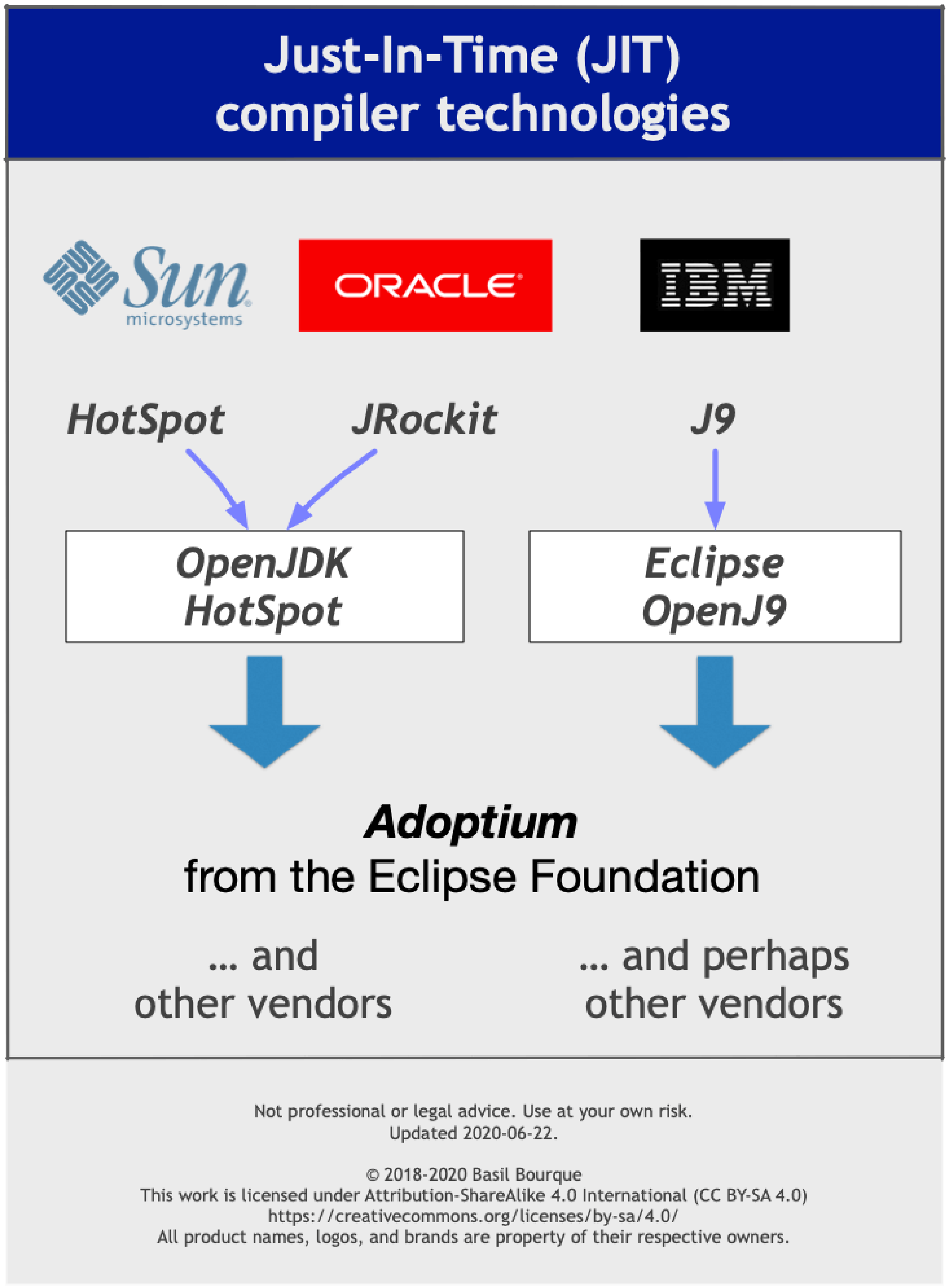
Java > jdk-21

Name	Date modified
bin	05-02-2024 17:52
conf	05-02-2024 17:52
include	05-02-2024 17:52
jmods	05-02-2024 17:52
legal	05-02-2024 17:52
lib	05-02-2024 17:52
LICENSE	05-02-2024 17:52
README	05-02-2024 17:52
release	05-02-2024 17:52

9 items 1 item selected

Java SE Java virtual machines (JVMs)

1. **Codename One**: Utilizes the open-source **ParparVM**.
2. **Eclipse OpenJ9**: An open-source JVM from IBM (J9) that runs on platforms like AIX, Linux (x86, Power, and Z), macOS, Windows, MVS, OS/400, Pocket PC, and z/OS.
3. **GraalVM**: Based on HotSpot/OpenJDK, it features a polyglot capability, allowing seamless mixing and matching of supported languages.
4. **HotSpot**: The open-source Java VM implementation by Oracle, widely used in Java development work.


Just-In-Time (JIT) compiler technologies
Sun microsystems
ORACLE
IBM
HotSpot
JRockit
J9
OpenJDK HotSpot
Eclipse OpenJ9
Adoptium from the Eclipse Foundation
... and other vendors
... and perhaps other vendors
Not professional or legal advice. Use at your own risk.
Updated 2020-06-22.
© 2018-2020 Basil Bourque
This work is licensed under Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)
https://creativecommons.org/licenses/by-sa/4.0/
All product names, logos, and brands are property of their respective owners.

Java version history

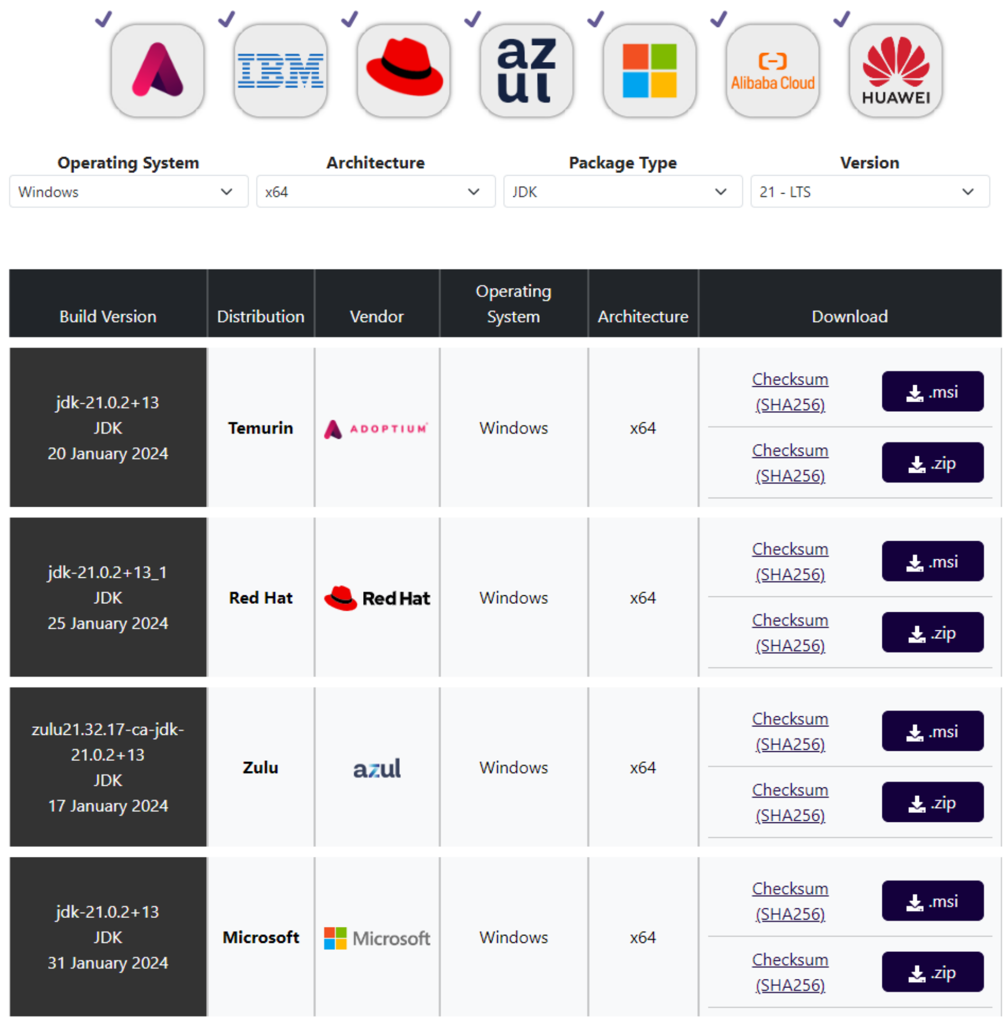
Version	Class File Format Version[1]	Release date	End of Public Updates (Free)	End of Extended Support (Paid)
JDK 1.0	45	23rd January 1996	May 1996	—
JDK 1.1	45	18th February 1997	October 2002	—
J2SE 1.2	46	4th December 1998	November 2003	—
J2SE 1.3	47	8th May 2000	March 2006	—
J2SE 1.4	48	13th February 2002	October 2008	—
J2SE 5.0	49	30th September 2004	October 2009	—
Java SE 6	50	11th December 2006	April 2013	December 2016 for Red Hat[2]October 2018 for Oracle[3]March 2026 for BellSoft Liberica[4]December 2027 for Azul[5]
Java SE 7	51	28th July 2011	July 2015	June 2020 for Red Hat[2]July 2022 for Oracle[6]March 2026 for BellSoft Liberica[4]December 2027 for Azul[5]
Java SE 8 (LTS)	52	18th March 2014	April 2019 for OracleJuly 2026 for Amazon Corretto[7]November 2026 for Eclipse Temurin[8]November 2026 for Red Hat[2]December 2030 for Azul[5]March 2031 for BellSoft Liberica[4]	December 2030 for Oracle[9]
Java SE 9	53	21st September 2017	March 2018	—
Java SE 10	54	20th March 2018	September 2018	—
Java SE 11 (LTS)	55	25th September 2018	April 2019 for OracleOctober 2024 for Eclipse	January 2032 for Oracle[9]

Version	Class File Format Version[1]	Release date	End of Public Updates (Free)	End of Extended Support (Paid)
			Temurin[8]October 2024 for Red Hat[2]March 2027 for BellSoft Liberica[4]October 2027 for Amazon Corretto[7]January 2032 for Azul[5]	
Java SE 12	56	19th March 2019	September 2019	—
Java SE 13	57	17th September 2019	March 2020	—
Java SE 14	58	17th March 2020	September 2020	—
Java SE 15	59	16th September 2020	March 2021	—
Java SE 16	60	16th March 2021	September 2021	—
Java SE 17 (LTS)	61	14th September 2021	September 2024 for Oracle[10]October 2027 for Eclipse Temurin[8]October 2027 for Red Hat[2]October 2028 for Amazon Corretto[7]September 2029 for Azul[5]March 2030 for BellSoft Liberica[4]	September 2029 for Oracle[9]
Java SE 18	62	22nd March 2022	September 2022	—
Java SE 19	63	20th September 2022	March 2023	—
Java SE 20	64	21st March 2023	September 2023	—
Java SE 21 (LTS)	65	19th September 2023	September 2026 for Oracle[10]September 2029 for Eclipse Temurin[8]September 2029 for Red Hat[2]October 2030 for Amazon	September 2031 for Oracle[9]

Version	Class File Format Version[1]	Release date	End of Public Updates (Free)	End of Extended Support (Paid)
			Corretto[7]September 2031 for Azul[5]March 2032 for BellSoft Liberica[4]	
Java SE 22	66	19th March 2024	September 2024	—

Top 6 Java Development Kit (JDK) Distributions

- Amazon Corretto.
- Azul Platform Core (Zulu)
- Oracle Java Downloads.
- Microsoft Build of OpenJDK.
- SAPMachine.
- Azul Platform Prime (Zing)



Operating System: Windows
Architecture: x64
Package Type: JDK
Version: 21 - LTS

Build Version	Distribution	Vendor	Operating System	Architecture	Download
jdk-21.0.2+13 JDK 20 January 2024	Temurin	ADOPTIUM	Windows	x64	Checksum (SHA256) .msi Checksum (SHA256) .zip
jdk-21.0.2+13_1 JDK 25 January 2024	Red Hat	Red Hat	Windows	x64	Checksum (SHA256) .msi Checksum (SHA256) .zip
zulu21.32.17-ca-jdk-21.0.2+13 JDK 17 January 2024	Zulu	azul	Windows	x64	Checksum (SHA256) .msi Checksum (SHA256) .zip
jdk-21.0.2+13 JDK 31 January 2024	Microsoft	Microsoft	Windows	x64	Checksum (SHA256) .msi Checksum (SHA256) .zip

Notable Java features introduced from version 8 to 21:

1. **Java 8**:
   - **Lambda Expressions**: Enables functional programming.
   - **Streams API**: Simplifies data manipulation.
   - **Default Methods**: Adds methods to interfaces.
   - **Optional**: Handles null values safely.
   - **New Date and Time API**: Improved date handling.
2. **Java 9**:
   - **Module System (Jigsaw)**: Enhances modularity.
   - **Private Interface Methods**: Allows private methods in interfaces.
   - **Process API Updates**: Provides better process management.

3. **Java 10**:
    - **Local Variable Type Inference (var)**: Simplifies variable declarations.
    - **Application Class-Data Sharing**: Optimizes startup time.
    - **Enhanced `Optional` Methods**: More utility methods.
4. **Java 11**:
    - **HTTP Client API**: Native HTTP client.
    - **Local-Variable Syntax for Lambda Parameters**: Cleaner lambda syntax.
    - `String::repeat` **Method**: Repeats strings easily.
5. **Java 12**:
    - **Switch Expressions (Preview)**: Enhanced `switch` statements.
    - `String::transform` **Method**: String manipulation.
    - `Collector::teeing` **Method**: Combines collectors.
6. **Java 13**:
    - **Text Blocks (Preview)**: Multiline string literals.
    - `switch` **Expressions Enhancements**: Simplified syntax.
    - `yield` **Statement (Preview)**: Used with `switch`.
7. **Java 14**:
    - **Records (Preview)**: Concise data classes.
    - **Pattern Matching for `instanceof` (Preview)**: Simplifies type checks.
    - `NullPointerException` **Improvements**: More informative messages.
8. **Java 15**:
    - **Sealed Classes (Preview)**: Restricts subclasses.
    - **Hidden Classes**: Enhances encapsulation.
    - **Text Blocks Enhancements**: Escaping and formatting.
9. **Java 16**:
    - **Records Enhancements**: Adds `equals`, `hashCode`, and `toString`.
    - **Pattern Matching for `switch` (Preview)**: Refines `switch`.
    - `sealed` **and** `non-sealed` **Classes**: Fine-grained access control.
10. **Java 17**:
    - **Sealed Classes (Finalized)**: Improved sealed types.
    - **Pattern Matching for `instanceof` (Finalized)**: Simplifies code.
    - **Foreign Function & Memory API (Incubator)**: Interacts with native code.

11. **Java 18** (Preview):
    - **Pattern Matching for `switch` (Finalized)**: Full support.
    - **Value Types (Preview)**: Efficient primitive-like objects.
    - **Pattern Matching for Records (Preview)**: Simplifies record handling.
12. **Java 19** (Preview):
    - **Pattern Matching for Records (Finalized)**: Complete record patterns.
    - `record` **Enhancements**: More flexibility.
    - `sealed` **and** `non-sealed` **Classes (Finalized)**: Refined access control.
13. **Java 20** (Preview):
    - **Pattern Matching for `switch` (Enhancements)**: Further improvements.
    - `record` **Enhancements**: Additional features.
    - `sealed` **and** `non-sealed` **Classes (Enhancements)**: Fine-tuning.
14. **Java 21** (Preview):
    - **Pattern Matching for `switch` (Further Enhancements)**: Continued refinements.
    - **Record Patterns for `switch` and `instanceof`**: Deconstruct complex structures.
    - **String Templates**: Extensible and safe string interpolation.

Reasons to Upgrade

Upgrading from **Java 8** to a more recent version like **Java 21** brings several benefits and enhancements. Here are **ten compelling reasons** to consider making the leap:

1. Security Enhancements: Java 21 includes security patches, bug fixes, and improvements that help safeguard your code against vulnerabilities and potential threats.

- Performance Boosts: Newer Java versions often come with performance optimizations, making your applications run faster and more efficiently.
- Language Features: Java 21 introduces exciting language features like pattern matching, records, and sealed classes. These features enhance code readability and expressiveness.
- Framework Compatibility: Staying up-to-date ensures compatibility with evolving frameworks and libraries. It allows you to leverage the latest features and improvements in popular frameworks like Spring Boot 3.
- API Improvements: Java 21 brings enhancements to existing APIs and introduces new ones. These updates can simplify your code and improve developer productivity.
- Module System: If your application is modular, migrating to Java 21 ensures compatibility with the Java module system introduced in Java 9.

- Dependency Updates: Update third-party dependencies to versions compatible with Java 21. This ensures you're using the latest and most secure libraries.
- Build Tool Updates: Keep your build tools (such as Maven or Gradle) up-to-date to support the target Java version.
- IDE Compatibility: Ensure your Integrated Development Environment (IDE) supports Java 21 for seamless development and debugging.
- Documentation and Testing: Update your documentation to reflect code changes and thoroughly test your application after migration to ensure it works as expected.

Challenges of Upgrading

- Compatibility issues with existing code can be a significant challenge during migration.
- Newer versions may not support certain features or syntax used in older versions.
- Deprecated APIs require developers to find suitable alternatives or refactor the code.
- Changes in default behaviour in newer Java versions can result in unexpected results or errors.
- Developers may need to adapt their code to accommodate these changes.
- Hibernate 6 is more strict and was in my migration projects the biggest pain
- javax to jakarta shall prove to be one of the most painful part of migration. The problem is that a lot of libraries are not yet migrated.
- The worst part of it is the amount of Spring Boot 2 content out there is enormous, and sometimes doesn't apply at all to Spring Boot 3.
-

Assessing Code Compatibility

- Before upgrading, assess the compatibility of your existing codebase with the target Java version.
- Check if the current code would run without issues on Java 17.
- Utilize tools such as the Java Development Kit's (JDK) `jdeps` utility.
- `jdeps` can help identify potential problems and offer guidance on resolving them.
- Consider using other conversion tools as well.

Handling Deprecated APIs

Java's evolution has seen certain APIs being deprecated and replaced with newer, more efficient alternatives. While migrating to Java 17, developers need to contend with these changes. Finding suitable replacements for deprecated APIs or refactoring the code to eliminate their use can be a complex but necessary task. This process is crucial to ensure that your applications continue to function efficiently and effectively in the newer Java environment.

JDK 9:

- Remove `apple.applescript` and `com.apple` packages.
- Disable X.509 certificate chains with SHA-1 based signatures.
- Remove launch-time JRE version selection directives.
- Remove the `jhat` tool.
- Remove JVM TI hprof agent.
- Remove GC combinations deprecated in JDK 8.
- Deprecate the Concurrent Mark Sweep Garbage Collector.
- Remove endorsed standards override and extensions mechanisms.

JDK 10:

- Remove the `javah` tool.

JDK 11:

- Remove Java EE.
- Remove CORBA.
- Remove `Thread.destroy()` and `Thread.stop(Throwable)`.
- The `var` keyword is no longer a valid class name.

JDK 14:

- Remove Solaris and SPARC ports.
- Disable biased locking by default and deprecate related flags.

JDK 15:

- Remove the Concurrent Mark Sweep (CMS) Garbage Collector.
- Deprecate the ParallelScavenge + SerialOld GC combination.
- Remove Pack200 tools and API, deprecate Pack200.
- Remove `rt.jar` from the JRE.

JDK 16:

- Remove the Nashorn JavaScript Engine.

JDK 17:

- Warn when `strictfp` modifier is used by default.
- Remove RMI activation (deprecated in JDK 15).

- Deprecate the Applet API (deprecated in JDK 9).
- Explore alternatives to the Security Manager.
- Remove experimental AOT and JIT compiler.
- Deprecate constructors of primitive wrapper classes.

JDK 18:

- Deprecate finalization and `Thread.stop`.
- Restrict JARs signed with SHA-1 algorithms.
- Strongly encapsulate internal APIs.
- Deprecate `java.net.URL` constructors.

JDK 21:

- Prepare to disallow dynamic loading of agents.

Changes in Default Behaviour

With each new version, Java may introduce changes in default behaviour that can lead to unexpected outcomes or even errors. It's therefore critical to conduct thorough testing to identify these issues. Once identified, the code may need to be adapted to accommodate these changes. This process, while potentially time-consuming, is crucial to ensuring a smooth transition to the new Java version.

Training Developers

Preparing developers for the upgrade by training them on the new features and syntax changes introduced in Java 17 is a critical aspect of the migration process, and should not be overlooked. This essential training involves a process of familiarisation with the new capabilities and modifications that Java 17 introduces, which can often be quite extensive and require a period of adjustment.

It's important to realise that the learning curve may be steep for some, but with the right resources and support, developers can quickly become proficient in the updated language. Fortunately, there are numerous resources available online to aid in this transition. These include the official Oracle documentation, which provides comprehensive details on the changes and new additions.

In addition to this, there are various tutorials available, which can provide a more interactive and practical way for developers to get to grips with Java 17. These tutorials often include examples and exercises that can help to solidify understanding and provide practical experience with using the new features.

Furthermore, developers can also benefit from online forums, where they can discuss challenges, share solutions, and gain insights from others who are also navigating the migration. These online communities can be a valuable source of support during this transition period.

These resources, when used effectively, can prove invaluable in helping developers get up to speed with the latest Java features. This in turn ensures a smooth and successful migration, allowing the team to fully utilise the new capabilities of Java 17 as soon as possible.

Testing and Deployment

In the context of any migration process, comprehensive testing emerges as an indispensable component. This involves a rigorous examination of various facets such as unit testing and integration testing, which play a pivotal role in maintaining the stability and robustness of the system. But the scope of testing extends beyond these basic checks. It is crucial to test the application in diverse environments, each potentially having unique configurations and conditions. The purpose of this broad-spectrum testing is to guarantee compatibility and ensure that the application behaves as expected, delivering consistent performance regardless of the specifics of the environment in which it is run.

Another vital aspect to consider during this process is the formulation of a detailed, well-thought-out deployment plan. Having such a plan in place can significantly mitigate the risk of unforeseen complications arising during the migration. It allows for the identification and rectification of potential issues before they become critical problems, and ensures the availability of contingency measures in case of unexpected circumstances. In essence, a comprehensive deployment plan plays a crucial role in facilitating a smooth, unimpeded transition to the new Java version, effectively minimising downtime and maximising efficiency.

Spring Boot Upgrade Tools

There are several tools like OpenRewrite, SBM and IntelliJ

Conclusion

The migration from Java 8 to Java 17 is a complex task that requires careful planning, rigorous testing, and possibly significant code changes. However, with a strategic approach and the right resources, the process can be made less daunting and ultimately rewarding. The key is to understand the challenges involved and prepare for them, ensuring a smooth and successful transition to the enhanced performance, security, and modern capabilities that Java 17 offers.

References

https://docs.oracle.com/en/java/javase/21/index.html

https://advancedweb.hu/a-categorized-list-of-all-java-and-jvm-features-since-jdk-8-to-21

https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2

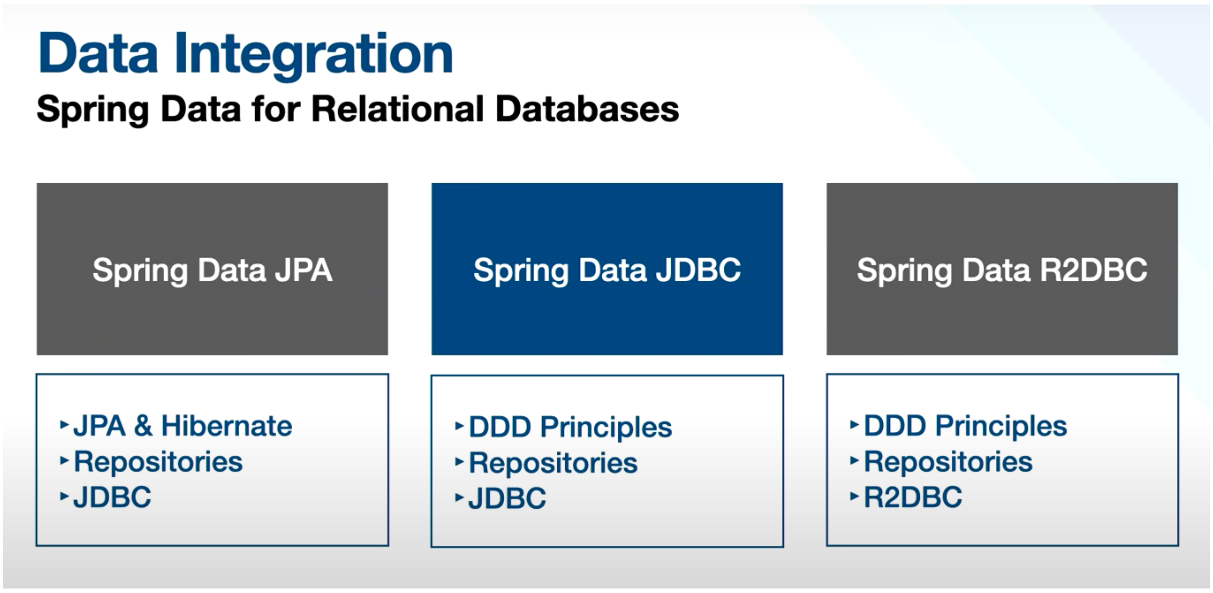
https://docs.openrewrite.org/recipes/java/spring/boot3/upgradespringboot_3_2

https://adoptium.net/en-GB/

https://www.eclipse.org/downloads/

https://docs.spring.io/spring-framework/reference/overview.html

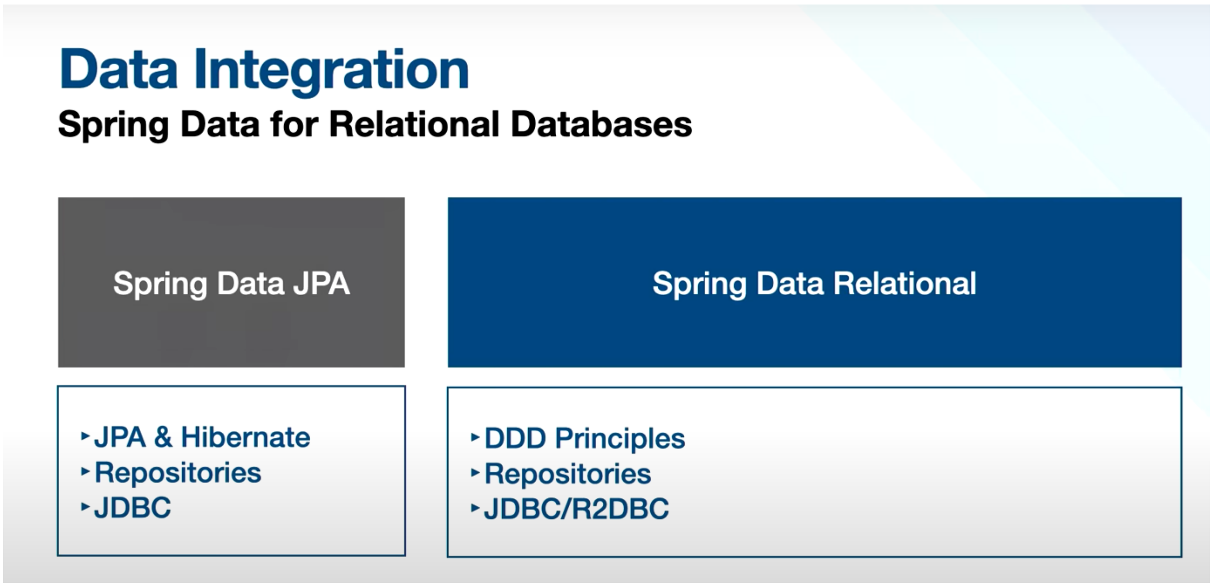
- **Introduction: Evolution of Java Versions**
- **Java Development Kit (JDK): Overview**
- **Java Development Kit (JDK) Version Changes**
- **Jakarta EE Introduction and its Impact**
- **Java Persistence API (JPA) 3.1: Enhancements and Migration**
- **Compatibility Issues: Java 8 vs. Java 21**
- **Migration Challenges and Best Practices**
- **Performance Optimization: Java 21 Benefits**
- **Developing with Java 21: Tools and Resources**
- **Migration Case Studies: Successful Transition Projects**
- **Key Considerations for Upgrading to Java 21**
- **Conclusion: Java 21 Migration Insights and Recommendations**



Data Integration

Spring Data for Relational Databases

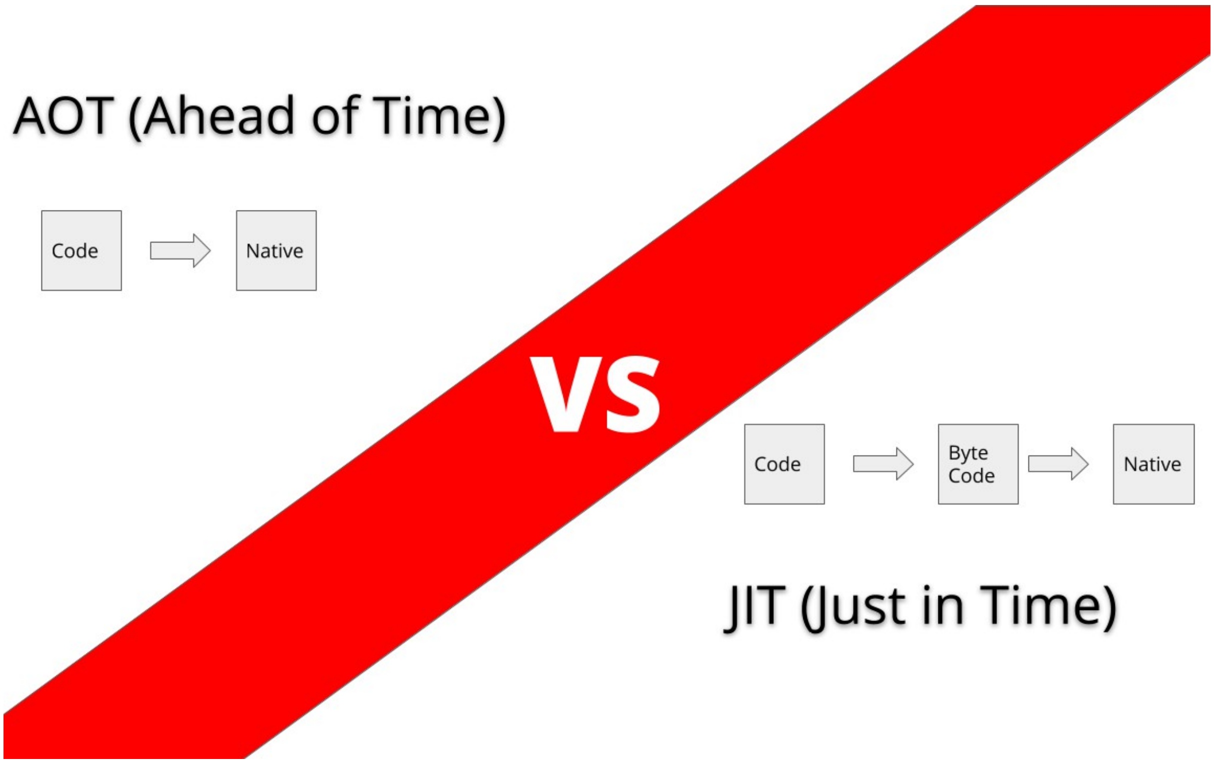
Spring Data JPA	Spring Data JDBC	Spring Data R2DBC
‣JPA & Hibernate ‣Repositories ‣JDBC	‣DDD Principles ‣Repositories ‣JDBC	‣DDD Principles ‣Repositories ‣R2DBC



Data Integration

Spring Data for Relational Databases

Spring Data JPA	Spring Data Relational
‣ JPA & Hibernate ‣ Repositories ‣ JDBC	‣ DDD Principles ‣ Repositories ‣ JDBC/R2DBC


AOT (Ahead of Time)
Code
Native
VS
Code
Byte Code
Native
JIT (Just in Time)