Java 12

Java 12 was released in March 2019 and introduced several enhancements and new features to improve developer productivity and application performance. While it is a non-LTS (Long-Term Support) release, it brought significant updates to the Java platform.

1. New String and Files Methods:

Java 12 introduces new methods in the String and Files classes, enhancing their functionality and usability.

String.indent(): The String.indent() method adjusts the indentation of each line in a string. It can add or remove spaces from the beginning of each line, providing a convenient way to format multi-line strings.

Explanation:

- **Indentation**: The indent() method takes an integer argument specifying the number of spaces to add or remove. A positive value adds spaces, while a negative value removes spaces.
- **Formatting**: This method is useful for formatting strings that span multiple lines, allowing developers to control indentation for better readability.

String.transform(): The String.transform() method applies a given function to a string and returns the result. It allows for chaining transformations and enhances the expressiveness of string manipulation.

Explanation:

- **Transformation**: The transform() method takes a function as an argument and applies it to the string, returning the result.
- **Chaining**: This method enables chaining of string transformations, allowing for more concise and expressive code.

Files.mismatch(): The Files.mismatch() method compares the contents of two files and returns the position of the first byte that differs. If the files are identical, it returns -1. This method provides a convenient way to compare files and identify differences.

Explanation:

- **Comparison**: The mismatch() method compares the contents of two files byte by byte and returns the position of the first mismatch.
- **Efficiency**: This method is efficient for comparing large files and quickly identifying differences.

2. The Teeing Collector:

The Teeing Collector is a new collector in the Collectors class that allows the collection of elements into two different results and then merges them into a final result. This feature enhances the capabilities of the Streams API, enabling more complex data processing in a single pass.

Explanation:

- **Teeing**: The teeing() method takes three arguments: two downstream collectors and a merging function. It collects elements into the two downstream collectors and merges the results using the merging function.
- **Single Pass**: The Teeing Collector allows for complex data processing in a single pass through the stream, improving performance and readability.

3. Support for Compact Number Formatting:

Java 12 introduces support for compact number formatting in the NumberFormat class, allowing numbers to be formatted in a more readable and concise way. This feature is particularly useful for displaying large numbers in a user-friendly format.

Explanation

- **Locale-Specific**: The getCompactNumberInstance() method returns a NumberFormat instance for compact number formatting, with results tailored to the specified locale.
- **Readability**: Compact number formatting improves the readability of large numbers by using abbreviations, such as "M" for millions and "L" for lakhs.

4. Performance Improvements

Java 12 includes several performance improvements, enhancing the efficiency and scalability of Java applications.

Default CDS Archives:

Java 12 improves the startup time and memory footprint of Java applications by introducing default Class Data Sharing (CDS) archives. These archives contain preprocessed class metadata, reducing the time and resources required to load classes at runtime.

- **CDS Archives**: Default CDS archives store metadata for common Java classes, allowing the JVM to load classes more efficiently and reduce memory usage.
- **Startup Time**: By using preprocessed metadata, default CDS archives improve the startup time of Java applications, especially those with large class hierarchies.

Abortable Mixed Collections for G1:

Java 12 enhances the G1 garbage collector by introducing abortable mixed collections, which allow garbage collection to be interrupted and resumed based on application needs. This feature reduces pause times and improves the responsiveness of applications using G1.

- **Mixed Collections**: G1 performs mixed garbage collection cycles, reclaiming both young and old-generation objects. Abortable mixed collections allow these cycles to be paused and resumed, reducing the impact on application performance.
- **Responsive GC**: By allowing garbage collection to be interrupted, G1 can adapt to the application workload and provide more responsive garbage collection.

Promptly Return Unused Committed Memory from G1:

Java 12 improves G1's ability to return unused committed memory to the operating system, reducing Java applications' memory footprint and improving resource utilization.

- **Memory Efficiency**: By promptly returning unused memory, G1 reduces the overall memory consumption of Java applications, improving performance on memory-constrained systems.
- **Resource Utilization**: This feature enhances resource utilization, allowing applications to use memory more efficiently and reduce memory waste.

5. Experimental and Preview Features:

Java 12 includes experimental and preview features that allow developers to explore new capabilities and provide feedback before they become standard.

Switch Expressions (Preview):

Java 12 introduces switch expressions as a preview feature, allowing the switch statement to be used as an expression. This feature provides a more concise and flexible way to handle multiple cases, reducing boilerplate code.

Explanation

- **Expression Syntax**: Switch expressions use the -> syntax to associate a case with a result, allowing for more concise and readable code.
- **Improved Readability**: By reducing boilerplate code, switch expressions improve the readability and maintainability of Java applications.

Shenandoah: A Low-Pause-Time Garbage Collector (Experimental)

Shenandoah is an experimental garbage collector introduced in Java 12. It is designed to provide low-pause-time garbage collection, making it suitable for applications with large heaps and high memory demands.

- **Low-Pause-Time GC**: Shenandoah minimizes pause times by performing most of its work concurrently with the application, reducing the impact of garbage collection on performance.
- **Large Heap Support**: Shenandoah is designed to scale with large heap sizes, making it suitable for memory-intensive applications.

6. Other Changes in Java 12

Java 12 includes several other changes that enhance the platform and improve performance.

Unicode 11

Java 12 supports Unicode 11, which includes additional characters, scripts, and emoji. This support enhances Java's ability to handle diverse character sets and languages, improving internationalization capabilities.

- **Expanded Character Set**: Unicode 11 support allows developers to work with a broader range of characters and languages, enabling applications to better serve global audiences.

- **Improved Internationalization**: By supporting the latest Unicode standard, Java 12 enhances its ability to handle international text and data, improving the usability and accessibility of applications worldwide.

Microbenchmark Suite

Java 12 includes a microbenchmark suite that provides a framework for benchmarking Java code. This suite helps developers measure the performance of their applications and identify areas for optimization.

- **Performance Measurement**: The microbenchmark suite provides tools for measuring the performance of Java applications, allowing developers to identify performance bottlenecks and optimize code.
- **Code Optimization**: By providing detailed performance metrics, the microbenchmark suite helps developers improve the efficiency and scalability of their applications.

JVM Constants API

Java 12 introduces the JVM Constants API, which provides a way to model and interact with the Java Virtual Machine (JVM) 's constant pool entries. This API enhances the flexibility and efficiency of working with constant pool entries.

- **Constant Pool Access**: The JVM Constants API provides a standardized way to access and manipulate constant pool entries, improving the efficiency and flexibility of Java applications.
- **Enhanced Performance**: The JVM Constants API provides a more flexible way to work with constant pool entries, contributing to improved application performance and resource utilization.

One AArch64 Port, Not Two

Java 12 consolidates the AArch64 ports, providing a single port for ARM-based architectures. This change simplifies the build process and improves the maintainability of the Java platform on ARM-based systems.

- **Simplified Build Process**: Java 12 simplifies the build and maintenance of the Java platform on ARM-based systems by consolidating the AArch64 ports.
- **Improved Maintainability**: A single AArch64 port simplifies the development and maintenance of Java applications on ARM-based architectures.

Conclusion

Java 12 introduces several key features and improvements, including new string and file methods, the Teeing Collector, and compact number formatting. These features enhance developer productivity, application performance, and the flexibility of the Java platform. By understanding and leveraging these features, developers can build more efficient and maintainable Java applications.

For a complete list of all changes in Java 12, refer to the official **Java 12 release notes.**