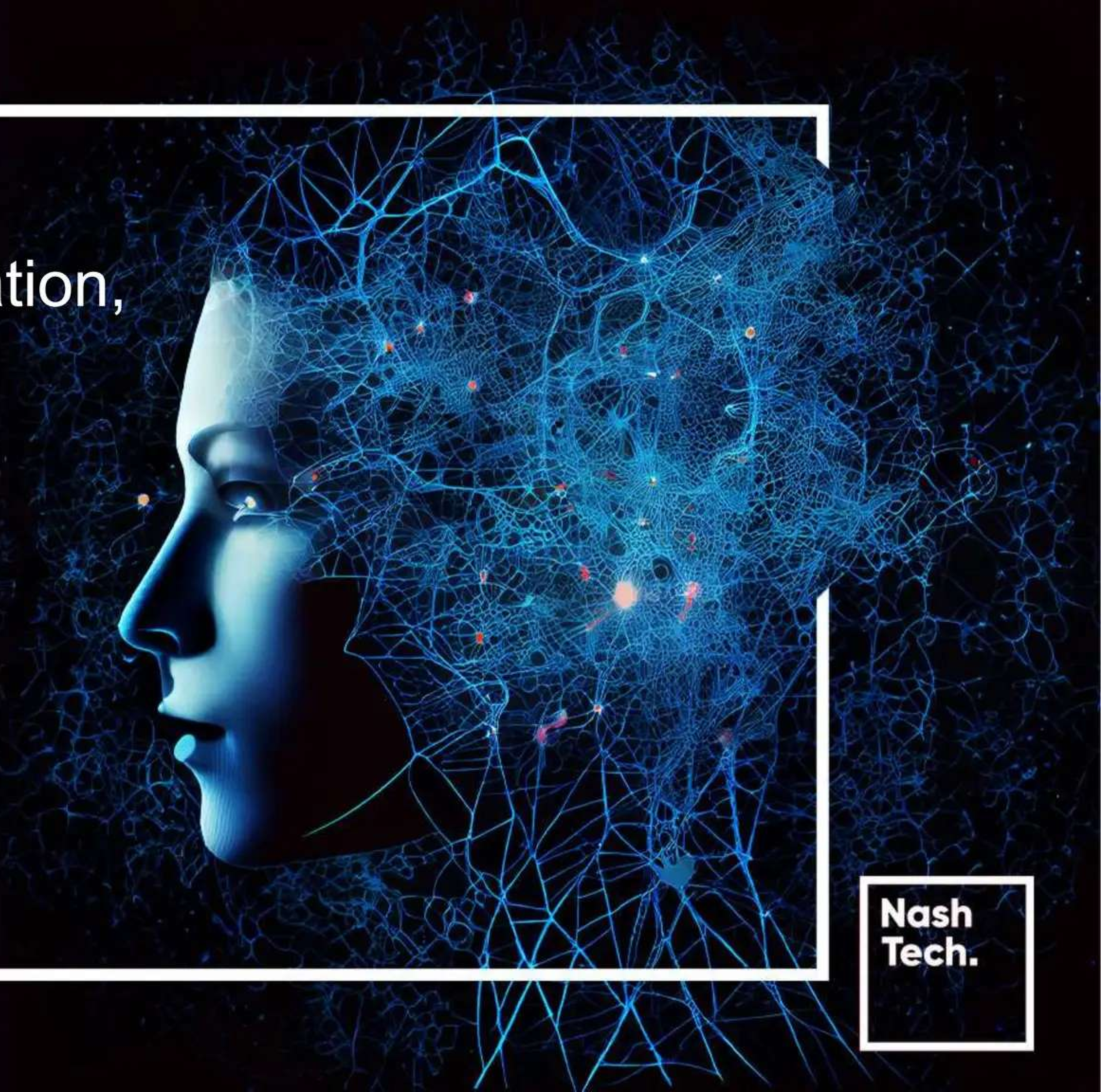
Java 17:
Features & implementation,



Chitra Singh (Senior Software Consultant)

Java Competency

KnolX Etiquettes

Lack of etiquette and manners is a huge turn off.

- **Punctuality**

  Join the session 5 minutes prior to the session start time. We start on time and conclude on time!

- **Feedback**

  Make sure to submit a constructive feedback for all sessions as it is very helpful for the presenter.

- **Silent Mode**

  Keep your mobile devices in silent mode, feel free to move out of session in case you need to attend an urgent call.

- **Avoid Disturbance**

  Avoid unwanted chit chat during the session.

Agenda

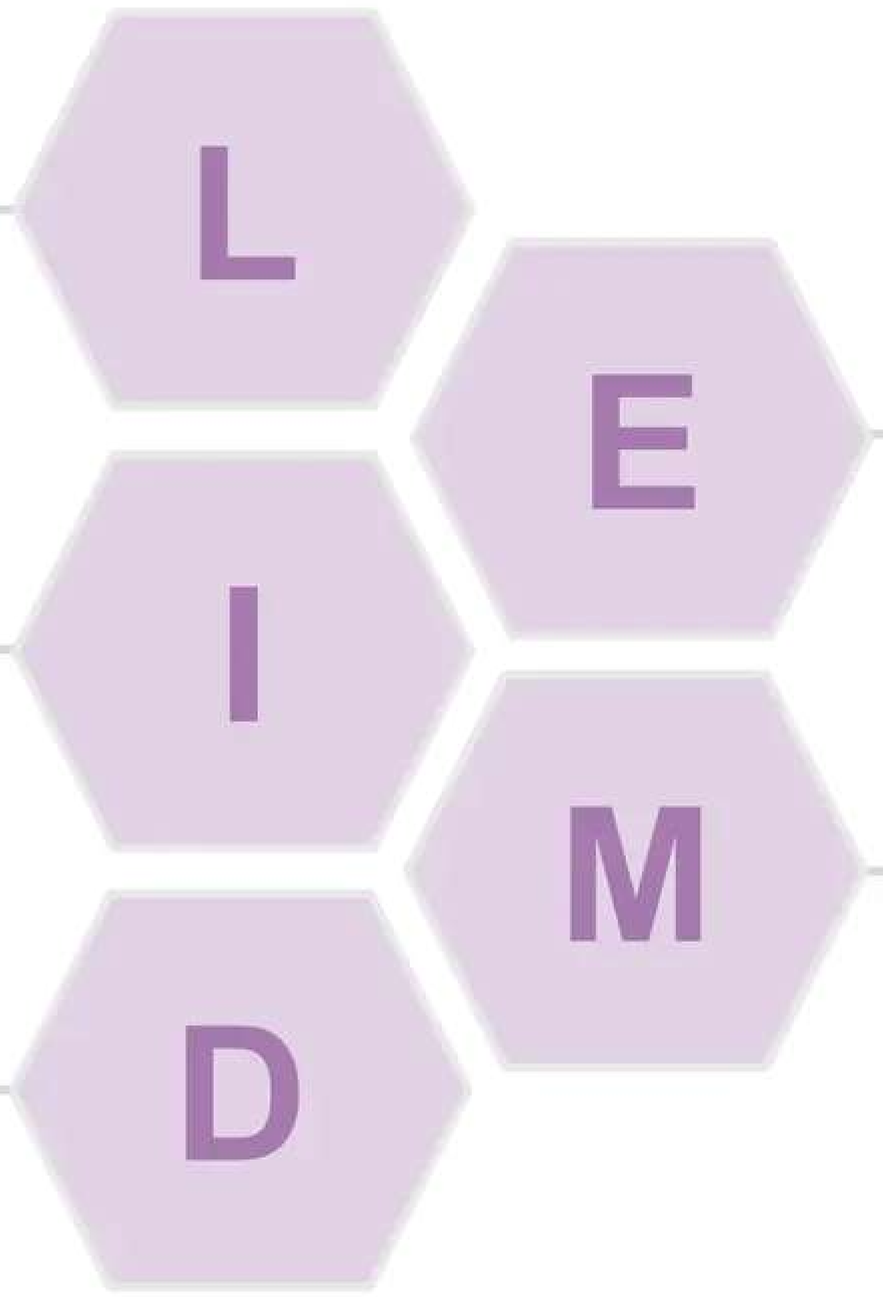
- **Introduction to Java 17**
  - Java 17 Overview
  - Key Highlights
  - Importance of Upgrading
- **Key New Features in Java 17**
  - Pattern Matching for switch (Preview)
  - Sealed Classes
  - Enhanced Pseudo-Random Number Generators
  - New Applet API Removal
  - Context-Specific Deserialization Filters
  - Foreign Function & Memory API (Incubator)
  - Vector API (Second Incubator)
  - Strong Encapsulation of JDK Internals by Default
  - Deprecate and Disable Biased Locking
  - macOS/AArch64 Port
- **QnA**

Introduction to Java 17

Java 17, released in September 2021, is a long-term support (LTS) release, which means it will receive extended support and updates from Oracle. This makes it a significant version for businesses and developers who prefer stability and long-term maintenance. Java 17 brings numerous enhancements, new features, and improvements that contribute to its performance, security, and overall functionality.

- **Java 17 Overview:**
  - Released in September 2021
  - Long-Term Support (LTS) version, supported until 2029
  - Successor to Java 11 LTS, following the 6-month release cadence
  - Part of Oracle's commitment to a predictable release schedule
- **Key Highlights:**
  - Enhanced performance and stability
  - Significant improvements in garbage collection and memory management
  - Introduction of new language features and APIs
- **Importance of Upgrading:**
  - Staying updated with the latest security patches and bug fixes
  - Leveraging new features for more efficient and readable code
  - Ensuring compatibility with modern libraries and frameworks

Benefits of Upgrading to Java 17



Long-Term Support
Java 17 is an LTS release, ensuring extended support and updates, making it a stable and reliable choice for production environments

Enhanced Performance
Includes performance improvements and optimizations that lead to better resource utilization and faster application execution

Improved Security
Strong encapsulation of JDK internals and context-specific deserialization filters enhance application security by reducing vulnerabilities

Modern Language Features
Features like sealed classes and pattern matching for switch enhance code readability, expressiveness, and maintainability

Developer Productivity
New APIs and language enhancements, such as the Vector API and Foreign Function & Memory API, provide powerful tools that increase developer efficiency and reduce development time

Key New Features in Java 17

1. Pattern Matching for switch (Preview)
2. Sealed Classes
3. Enhanced Pseudo-Random Number Generators
4. New Applet API Removal
5. Context-Specific Deserialization Filters
6. Foreign Function & Memory API (Incubator)
7. Vector API (Second Incubator)
8. Strong Encapsulation of JDK Internals by Default
9. Deprecate and Disable Biased Locking
10. macOS/AArch64 Port

- A **preview feature** is a new feature of the Java language, Java Virtual Machine, or Java SE API that is fully specified, fully implemented, and yet impermanent. It is available in a JDK feature release to provoke developer feedback based on real world use; this may lead to it becoming permanent in a future Java SE Platform.

- A **final feature** is a new feature of the Java language, Java Virtual Machine, or Java SE API that is fully specified, fully implemented, and permanent.

- **Incubator** modules are a means of putting non-final APIs and non-final tools in the hands of developers, while the APIs/tools progress towards either finalization or removal in a future release.

Features in Java 17(Pattern Matching for switch [Preview])

Pattern Matching for Switch (Preview)

Pattern Matching for switch is an enhancement in Java 17 that allows you to use patterns directly in switch statements and expressions. This feature simplifies complex conditional logic and enhances code readability and maintainability.

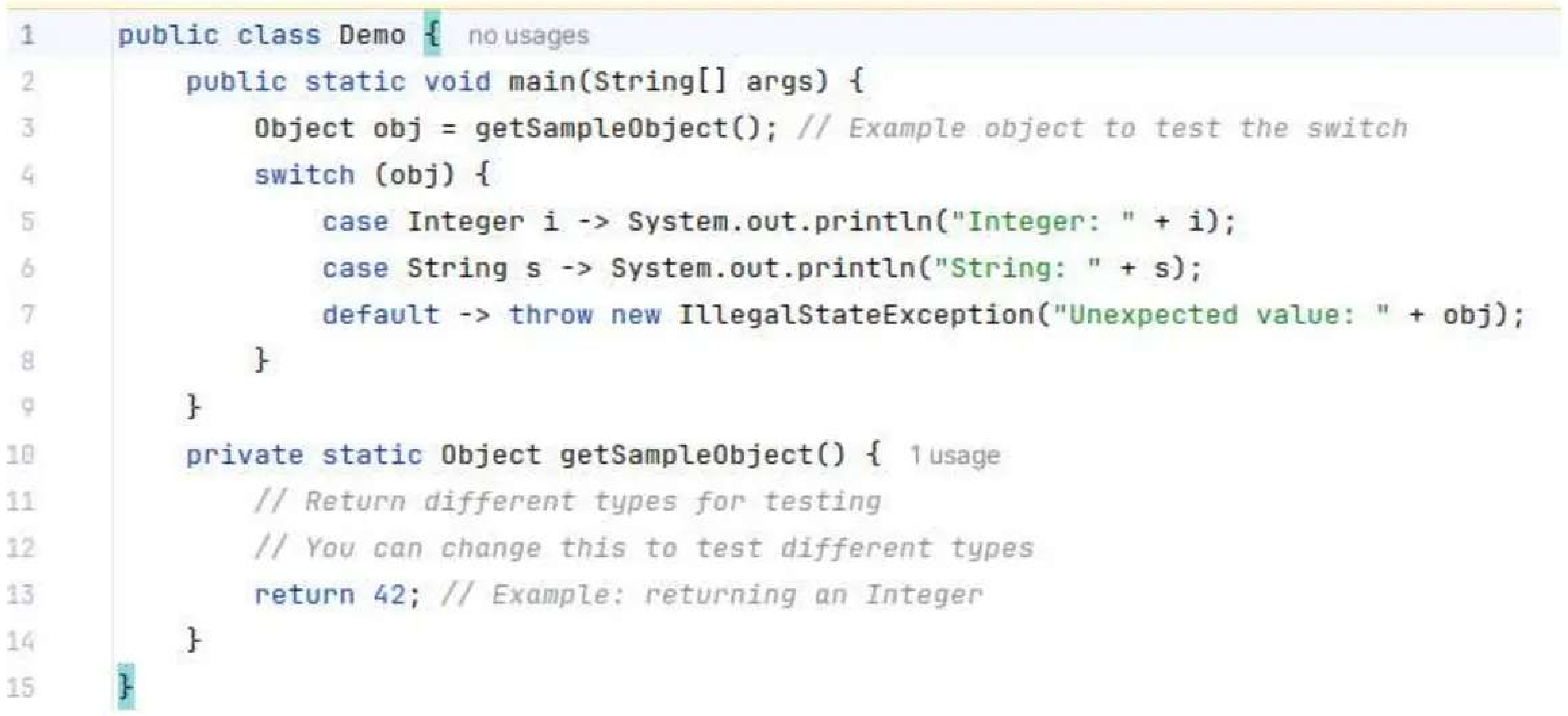
- Simplifies the syntax of complex switch statements
- Allows patterns to be used directly within switch statements and expressions

Benefits:

- More readable and concise code
- Enhanced type safety by ensuring that all cases are covered
- Reduces boilerplate code associated with type checks and casts

Features in Java 17(Pattern Matching for switch [Preview])

Implementation of Pattern Matching for Switch (Preview)



```java
public class Demo {  no usages
    public static void main(String[] args) {
        Object obj = getSampleObject(); // Example object to test the switch
        switch (obj) {
            case Integer i -> System.out.println("Integer: " + i);
            case String s -> System.out.println("String: " + s);
            default -> throw new IllegalStateException("Unexpected value: " + obj);
        }
    }
    private static Object getSampleObject() {  1 usage
        // Return different types for testing
        // You can change this to test different types
        return 42; // Example: returning an Integer
    }
}
```

Use Cases:

- Handling different types of objects in a collection
- Simplifying the processing logic based on the type of input
- Improving the clarity of conditional logic by reducing nested if-else statements

Features in Java 17 (Sealed Classes)

Sealed Classes

Sealed classes and interfaces are a powerful feature introduced in Java 17. They allow you to restrict which classes or interfaces can extend or implement them. This feature provides a way to create a more controlled and predictable class hierarchy.

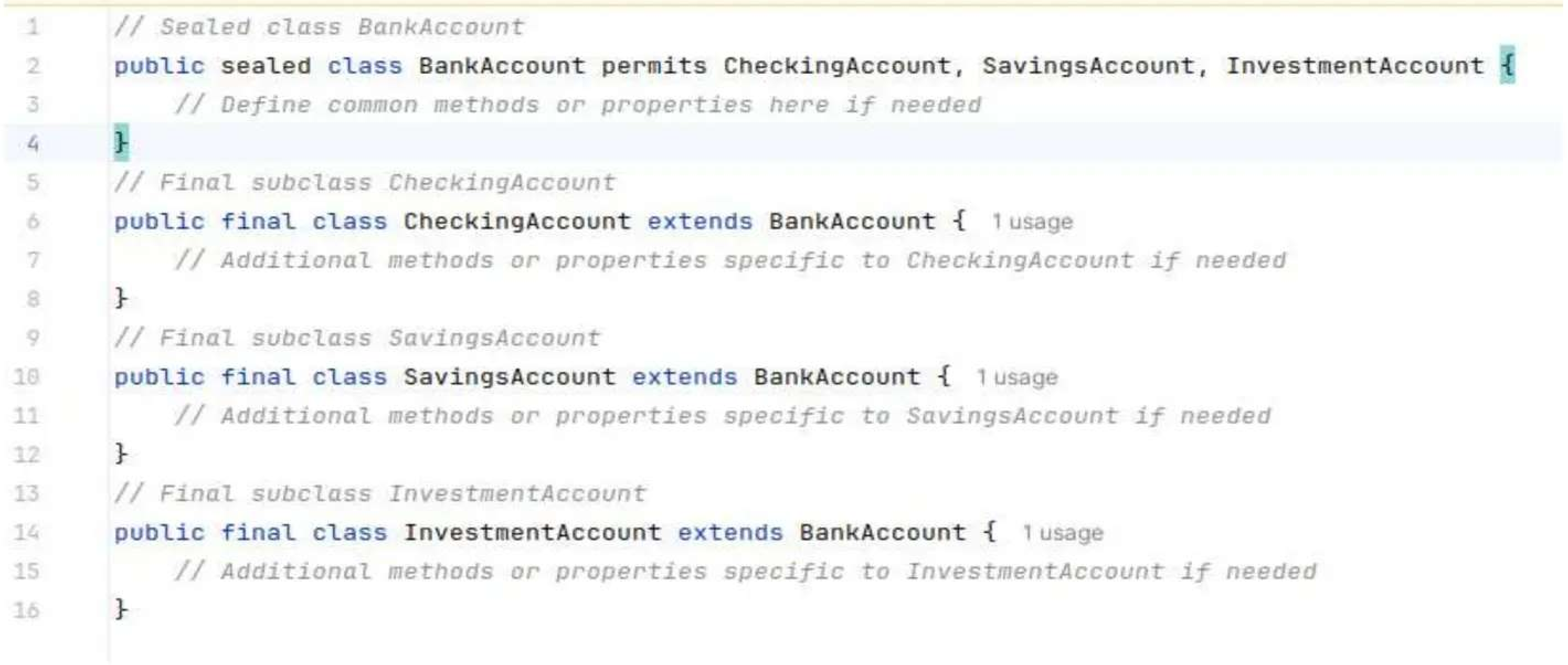
- Restrict which other classes or interfaces may extend or implement them
- Enhance the modeling of class hierarchies and provide better control over inheritance

Benefits:

- Ensures a limited set of subclasses, which can be easier to manage and understand
- Improves code readability and maintainability by explicitly specifying permitted subclasses
- Enables more secure and reliable designs by preventing unintended extensions

Features in Java 17 (Sealed Classes)

Implementation of Sealed Classes



```java
// Sealed class BankAccount
public sealed class BankAccount permits CheckingAccount, SavingsAccount, InvestmentAccount {
    // Define common methods or properties here if needed
}
// Final subclass CheckingAccount
public final class CheckingAccount extends BankAccount {  1 usage
    // Additional methods or properties specific to CheckingAccount if needed
}
// Final subclass SavingsAccount
public final class SavingsAccount extends BankAccount {  1 usage
    // Additional methods or properties specific to SavingsAccount if needed
}
// Final subclass InvestmentAccount
public final class InvestmentAccount extends BankAccount {  1 usage
    // Additional methods or properties specific to InvestmentAccount if needed
}
```

Use Cases:

- Defining a fixed set of types for a particular domain (e.g., different kinds of shapes, commands, etc.)
- Enhancing security by limiting which classes can extend a base class
- Providing a clear and enforced API for developers using the class

Features in Java 17 (Enhanced Pseudo-Random Number Generators)

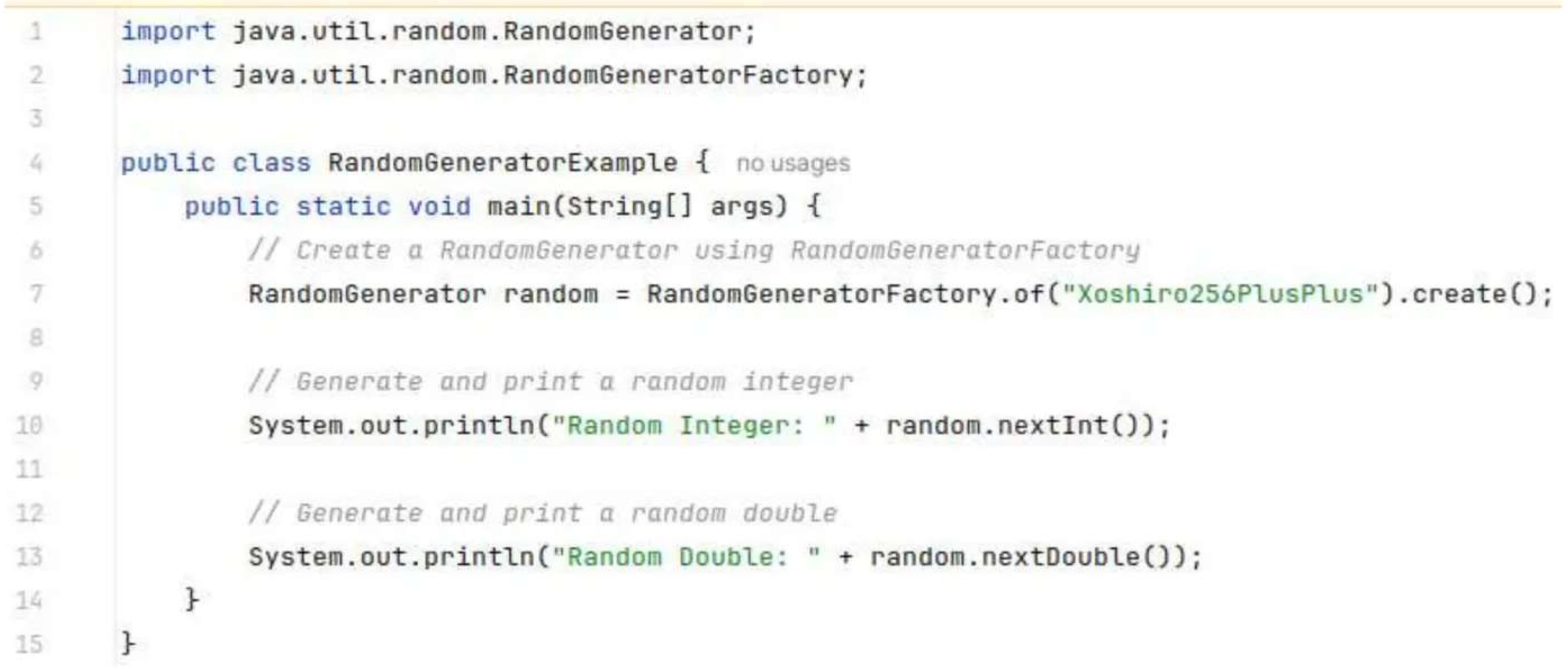
Enhanced Pseudo-Random Number Generators

Java 17 introduces several enhancements and features related to random number generation, building upon the existing ***java.util.Random*** and ***java.util.concurrent.ThreadLocalRandom*** classes. Here are some of the key enhancements related to pseudo-random number generation in Java 17:

- **New Interface: RandomGenerator** Abstraction for different random number generators.
- **SplittableRandom Implementation** Efficient generation of random numbers across threads. It Supports various algorithms and distributions.
- **Enhanced SecureRandom** Supports ThreadLocalRandom.current() for concurrency.
- **Cryptographically** strong randomness seeding.
- **Control and Flexibility APIs** Manage and configure random generators.
- **Performance Improvements** Enhanced algorithms for faster random number generation.

Features in Java 17 (Enhanced Pseudo-Random Number Generators)

Implementation of Enhanced Pseudo-Random Number Generators



```java
import java.util.random.RandomGenerator;
import java.util.random.RandomGeneratorFactory;

public class RandomGeneratorExample {  no usages
    public static void main(String[] args) {
        // Create a RandomGenerator using RandomGeneratorFactory
        RandomGenerator random = RandomGeneratorFactory.of("Xoshiro256PlusPlus").create();

        // Generate and print a random integer
        System.out.println("Random Integer: " + random.nextInt());

        // Generate and print a random double
        System.out.println("Random Double: " + random.nextDouble());
    }
}
```

Use Cases:

- **Simulation:** Accurate modeling of complex systems.
- **Security:** Cryptographic operations and secure communication protocols.
- **Gaming:** Random event generation in gaming applications.
- **Analytics:** Statistical analysis and data sampling.

Features in Java 17 (New Applet API Removal)

Deprecate the Applet API for Removal

In JDK 17, the Applet API is marked as deprecated for removal. It means that the JDK 18 will remove the API.The Applet API removal is long overdue, considering that all browsers have either ended the support or have plans to do so. The API had already been deprecated in JDK 9, but it had not been marked for removal.

- **Applet API:** Historically used for embedding Java applets in web browsers.
- **Replacement:** HTML5, JavaScript, and modern web technologies now fulfill similar functionalities.

Reasons:

- Security vulnerabilities
- Decreased usage
- Modern web standards led to this decision

Implications:

- **End of Life:** Applets are no longer supported for running in web browsers as of Java 11 and beyond.
- **Transition:** Developers and organizations must migrate existing applet-based applications to alternative technologies.

Features in Java 17 (Context-Specific Deserialization Filters)

Context-Specific Deserialization Filters

Context-Specific Deserialization Filters provide a mechanism to control and restrict the deserialization process of objects based on contextual information. This is particularly useful in scenarios where deserialization of arbitrary objects from untrusted sources could pose security risks.

Key Features:

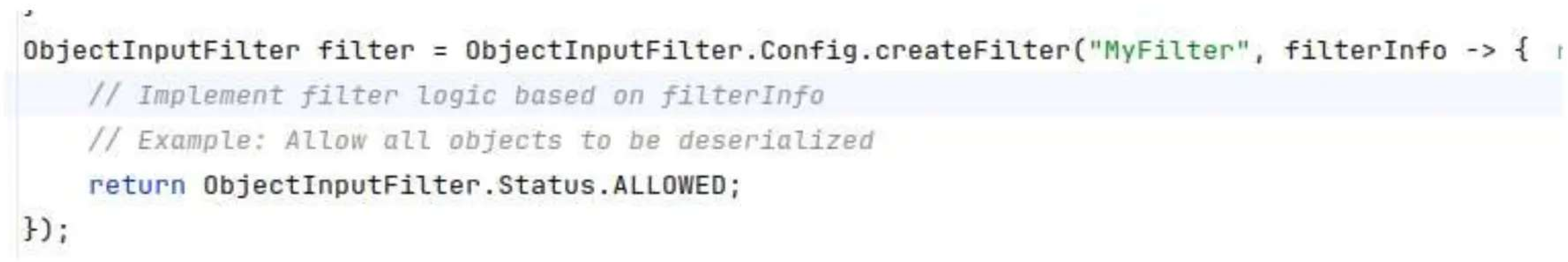
- **Filter Interface:** *java.io.ObjectInputFilter* provides methods to control deserialization behavior.
- **Contextual Criteria:** Define filters based on caller class, code source, or specific attributes.
- **Granular Control:** Allows per-call, per-thread, or per-application filters.

Benefits:

- **Security Enhancement:** Prevents deserialization attacks by limiting allowed object types.
- **Flexibility:** Customize filters based on specific application requirements.
- **Compatibility:** Integrates with existing deserialization mechanisms in Java.

Features in Java 17 (Context-Specific Deserialization Filters)

Implementation of Context-Specific Deserialization Filters



```
ObjectInputFilter filter = ObjectInputFilter.Config.createFilter("MyFilter", filterInfo -> {
    // Implement filter logic based on filterInfo
    // Example: Allow all objects to be deserialized
    return ObjectInputFilter.Status.ALLOWED;
});
```

Use Cases:

•**Network Communication:** Apply stricter filters for deserialization over network protocols.

•**Sensitive Data Handling:** Control deserialization in environments handling sensitive information.

•**Third-Party Libraries:** Protect against vulnerabilities in third-party library deserialization.

Features in Java 17 (Strong Encapsulation of JDK Internals by Default)

Strong Encapsulation of JDK Internals by Default

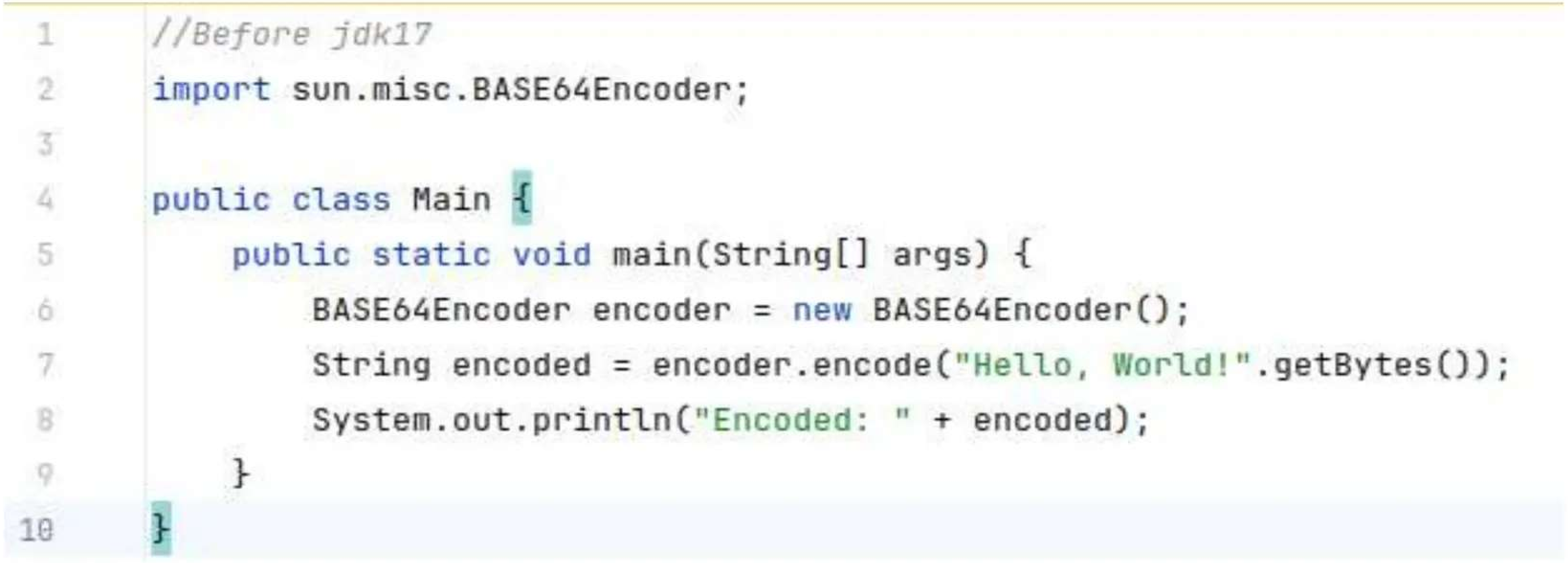
In Java 17, a significant enhancement was introduced regarding the strong encapsulation of JDK internals by default. This change aims to improve the security and maintainability of Java applications by restricting direct access to internal APIs that are not part of the public Java API specification.

- **Access Restriction**: Access to internal JDK classes and APIs (those in packages like ***sun.*, com.sun.*, jdk.internal.****, etc.) is restricted by default. Attempting to access these APIs from code outside the JDK itself will result in compilation errors or runtime exceptions.
- **Impact on Developers**:
  - **Migration**: Developers are required to migrate their code away from using internal APIs to public APIs or supported alternatives.
  - **Security**: This change enhances the security of Java applications by reducing the attack surface exposed through potentially insecure internal APIs.
  - **Maintainability**: Encourages developers to write more robust and maintainable code by relying only on stable, documented public APIs.
- **Migration Strategies**:
  - **Use Public APIs**: Identify and use public APIs or alternative libraries that provide equivalent functionality.
  - **Module System**: Utilize Java's module system (introduced in Java 9) to encapsulate internal implementations and expose only necessary public interfaces.
  - **Compatibility**: Use tools like the jdeps tool to analyze dependencies and identify usage of internal APIs that need migration.

Features in Java 17 (Strong Encapsulation of JDK Internals by Default)

Implementation of Strong Encapsulation of JDK Internals by Default

In Java 17, direct access to sun.misc.BASE64Encoder is restricted due to strong encapsulation of JDK internals. Therefore, you should use the java.util.Base64 class from the standard Java API instead. Here's how you can rewrite your code using java.util.Base64:



```
//Before jdk17
import sun.misc.BASE64Encoder;

public class Main {
    public static void main(String[] args) {
        BASE64Encoder encoder = new BASE64Encoder();
        String encoded = encoder.encode("Hello, World!".getBytes());
        System.out.println("Encoded: " + encoded);
    }
}
```

Features in Java 17 (Deprecate and Disable Biased Locking)

Deprecate and Disable Biased Locking:

Biased Locking Optimization for single-threaded access to locks.
As of Java 17, biased locking has already been deprecated and disabled in previous versions (Java 15 and 16).
Therefore, there are no additional changes or actions needed specifically for Java 17 regarding biased locking.

Why Deprecate and Disable?

- **Performance considerations**: Overhead in high-contention scenarios.
- **Simplification of JVM internals**: Enhances predictability and scalability.
- **Deprecation (Java 15)**:Officially deprecated due to Increased complexity. Limited Simplify JVM internals for better predictability and scalability.
- **Disablement (Java 16)** :Fully disabled to Remove overhead in high-contention scenarios. Simplify JVM internals for better predictability and scalability.

Alternative Strategies:

- Adaptive spinning, lock elision: Modern JVM optimizations for concurrency.
- Ensures better performance across diverse workloads.

Features in Java 17 (macOS/AArch64 Port)

macOS/AArch64 Port

- **Transition to Apple Silicon**
  - **Apple Silicon**: ARM-based processors (M1, M1 Pro, M1 Max) introduced by Apple for Macs.
  - **Performance**: Improved power efficiency and performance compared to Intel x86.

- **Java 17 Support**
  - **Native Compilation**: Java 17 supports AArch64 (ARM64) architecture natively.
  - **Compatibility**: Ensures Java applications run efficiently on macOS devices with Apple Silicon.

- **Development Ecosystem**
  - **Xcode Integration**: Java development using Xcode on macOS AArch64.
  - **Universal Binaries**: Facilitates deployment of Java applications as universal binaries for both ARM and x86 architectures.

- **Benefits**
  - **Performance Boost**: Native execution on Apple Silicon enhances Java application performance.
  - **Ecosystem Integration**: Enables developers to leverage new hardware capabilities seamlessly.

Features in Java 17 (Foreign Function & Memory API (Incubator),Vector API (Second Incubator))

Foreign Function & Memory API (Incubator)

Facilitates seamless interaction with native code and efficient management of native memory directly from Java.

Key Features

- **Native Library Interaction**: Enables calling native functions directly from Java without JNI (Java Native Interface) overhead.
- **Memory Access**: Provides safe and efficient access to native memory regions from Java code.
- **Type Safety**: Ensures type safety when interacting with native data structures.

Vector API (Second Incubator)

The Vector API (Second Incubator) in Java 17 introduces a set of features aimed at enabling efficient SIMD (Single Instruction, Multiple Data) programming in Java

Key Features

- **SIMD Parallelism**: Enables efficient parallel processing of data using vector operations.
- **Performance**: Enhances computational throughput for numerical and data-intensive tasks.
- **Platform Support**: Designed to leverage hardware SIMD support for optimized execution.

Thank you

Nash Tech.