**Note –**The latest stable release of Java is Java 17. Java 17 was released on September 17, 2021 and is the latest long-term support (LTS) release.

Advertisement

Java 18, 19 are also available, but these releases are not long-term support releases and are intended for early adopters and developers who want to try out new features and APIs.

JAVA 20 and JAVA 21 – See last line of this page-

Java 7 (released in 2011)

| **Feature** | **Java 7** | **Prior to Java 7** |
| --- | --- | --- |
| Binary literals | int binary = 0b10101010; | Not supported |
| Improved type inference | Map<String, List<Integer>> map = new HashMap<>(); | Map<String, List<Integer>> map = new HashMap<String, List<Integer>>(); |
| Multiple exception handling | `try {…} catch (IOException |SQLException ex) {…}` | NA |
| Automatic resource management | try (BufferedReader br = new BufferedReader(new FileReader("file.txt"))) {...} catch (IOException ex) {...} | BufferedReader br = new BufferedReader(new FileReader("file.txt")); try {...} finally {br.close();} |

Java 8 (released in 2014)

**1-** In Java, you can use lambda expressions to create implementation of functional interfaces. A functional interface is an interface with a single abstract method. Here is an example of a lambda expression in Java:

// Functional interface

@FunctionalInterface

interface MultiInter {

int multiply(int x, int y);

}

public class Main {

public static void main(String[] args) {

// Lambda expression to multiply two numbers

MultiInter multi = (int x, int y) -> x \* y;

// Call the lambda function

int result = multi.multiply(4, 5);

System.out.println(result); // Output: 20

}

}

In this example, the functional interface MultiInter has a single abstract method multiply that takes two int arguments and returns an int. The lambda expression (int x, int y) -> x \* y implements this abstract method and defines the logic for multiplying two numbers. The lambda expression is assigned to the functional interface MultiInter , and the lambda function can be called by calling the multiply method on the interface.

Advertisement

**2-**Stream API: The Stream API is a set of methods that allow you to perform operations on collections of data in a functional style. For example, the following code uses the Stream API to find the first even number in a list:

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);  
Optional<Integer> firstEven = numbers.stream()  
.filter(x -> x % 2 == 0)  
.findFirst();  
System.out.println(firstEven.get()); // prints 2  
  
**3-** In Java, a method reference is a way to refer to a method without actually invoking it. Method references can be used to pass methods as arguments to other methods, or to construct new functions from existing methods.  
  
Here is an example of a method reference in Java 8:  
  
import java.util.Arrays;  
import java.util.List;  
  
public class MethodReferenceExample {  
public static void main(String[] args) {  
List<String> strings = Arrays.asList("a", "b", "c", "d");  
  
// Use a lambda expression to print all elements in the list  
strings.forEach(s -> System.out.println(s));  
  
// Use a method reference to print all elements in the list  
strings.forEach(System.out::println);  
}  
}  
In this example, the forEach method of the List interface takes a lambda expression as an argument. The lambda expression specifies that for each element in the list, it should be printed to the console. The second call to forEach uses a method reference to the println method of the System.out object instead of a lambda expression.  
  
Both of these calls to forEach will have the same effect, printing all elements of the strings list to the console.

| **Feature** | **Java 8** | **Prior to Java 8** |
| --- | --- | --- |
| Lambda expressions | (x, y) -> x + y | N/A |
| Method references | System.out::println | N/A |
| Default methods | default void foo() {…} | N/A |
| New Date and Time API | LocalDate date = LocalDate.of(2020, Month.JANUARY, 1); | Calendar cal = Calendar.getInstance(); cal.set(2020, 0, 1); Date date = cal.getTime(); |
| Base64 Encoding | Base64.getEncoder(). encodeToString(str.getBytes()) | N/A |
| Stream API | list.stream().filter(s -> s.length() > 3).forEach(System.out::println); | N/A |

Java 9 (released in 2017)

| **Feature** | **Java 9** | **Prior to Java 9** |
| --- | --- | --- |
| Private methods in interfaces | private void foo() {} | N/A |
| Enhanced try-with-resources | try (resource) {} | try {} finally {} |
| Factory methods for ImmutableList, etc. | List.of(1, 2, 3) | Arrays.asList(1, 2, 3) |
| Collection.toArray(IntFunction) | list.toArray(String[]::new) | list.toArray(new String[list.size()]) |
| jshell – The Java Shell | Jshell | N/A |
| HTTP/2 Client | HttpClient client = HttpClient.newHttpClient(); | N/A |
| Modules (Project Jigsaw) | module com.example.app {} | N/A |

Advertisement

Java 10 (released in 2018)

| **Feature** | **Java 10** | **Prior to Java 10** |
| --- | --- | --- |
| Local-variable type inference | var num = 10; | int num = (int) 10; |
| var in lambda expressions | (var x, var y) -> x + y | (Integer x, Integer y) -> x + y |
| HTTP Client | var client = HttpClient.newHttpClient(); | HttpURLConnection connection = (HttpURLConnection) new URL(url).openConnection(); |

Java 11 (released in September 2018)

| **Feature** | **Java 11** | **Prior to Java 11** |
| --- | --- | --- |
| Local variable type | var list = new ArrayList<String>(); | ArrayList<String> list = new ArrayList<String>(); |
| HTTP Client | HttpClient client = HttpClient.newHttpClient(); | URL url = new URL(“[http://example.com&#8221](http://example.com&/#8221);); HttpURLConnection con = (HttpURLConnection) url.openConnection(); |
| File handling | String content = Files.readString(Paths.get(“file.txt”)); | BufferedReader reader = new BufferedReader(new FileReader(“file.txt”)); String line; while ((line = reader.readLine()) != null) { … } reader.close(); |
| Single-line if | if (bool) foo(); else bar(); | if (bool) { foo(); } else { bar(); } |
| Single-line for | for (int i : List.of(1, 2, 3)) foo(i); | for (int i : new int[] {1, 2, 3}) { foo(i); } |

Java 12 (released in March 2019)

| **Feature** | **Java 12** | **Prior to Java 12** |
| --- | --- | --- |
| Switch expressions | switch (day) { case MONDAY: return “Monday”; case TUESDAY: return “Tuesday”; default: return “Other day”; } | if (day == MONDAY) return “Monday”; else if (day == TUESDAY) return “Tuesday”; else return “Other day”; |
| Text blocks | String html = “”” <html> <body> <p>Hello, World</p> </body> </html> “””; | String html = “<html>\n” + ” <body>\n” + ” <p>Hello, World</p>\n” + ” </body>\n” + “</html>\n”; |
| Additional Unicode language support | System.out.println(“\uD83D\uDE02”); | System.out.println(“😂”); |

Java 13 (released in September 2019)

| **Feature** | **Java 13** | **Prior to Java 13** |
| --- | --- | --- |
| Dynamic CDS archives | java -XX:ArchiveClassesAtExit=myarchive.jsa | N/A |
|  | Creates a class data-sharing (CDS) archive at JVM exit for faster startup in future JVM runs |  |
| ZGC on macOS | java -XX:+UseZGC | N/A |
|  | Enables the Z Garbage Collector (ZGC) on macOS |  |
| Reimplementation of legacy DatagramSocket API | java.net.DatagramSocket | sun.net.DatagramSocketImpl |
|  | Replaces the legacy sun.net.DatagramSocketImpl implementation with a new implementation in the java.net package |  |
| Improved AARCH64 intrinsics | java -XX:+UseAARCH64Intrinsics | N/A |
|  | Enables intrinsic functions for the AARCH64 architecture to improve performance |  |
| Epsilon garbage collector | java -XX:+UseEpsilonGC | N/A |
|  | Enables the Epsilon garbage collector, which does not perform any actual memory reclamation and is intended for testing and benchmarking purposes |  |

Advertisement

Java 14 (released in March 2020)

| **Feature** | **Java 14** | **Prior to Java 14** |
| --- | --- | --- |
| Record classes | record Point(int x, int y) {} | N/A |
| Pattern matching for instanceof | if (obj instanceof Point p) {} | if (obj instanceof Point) { Point p = (Point) obj; } |
| Helpful NullPointerExceptions | NullPointerException.getMessage() | N/A |

Java 15 (released in September 2020)

| **Feature** | **Java 15** | **Prior to Java 15** |
| --- | --- | --- |
| Sealed classes | sealed class Shape permits Circle, Rectangle {} | N/A |
| Pattern matching for instanceof | if (obj instanceof Point p) {} | if (obj instanceof Point) { Point p = (Point) obj; } |

Java 16 (released in March 2021)

| **Feature** | **Java 16** | **Prior to Java 16** |
| --- | --- | --- |
| Records (Preview) | record Point(int x, int y) {} | N/A |
| Packaging tool | jpackage | N/A |
| Second Preview of Foreign-Memory Access API | java.lang.System.memory() | N/A |

Java 17 (released in September 2021)

| **Feature** | **Java 17** | **Prior to Java 17** |
| --- | --- | --- |
| Yield in switch statement | switch (day) { case MONDAY: yield TUESDAY; case FRIDAY: yield SUNDAY; } | N/A |
| Records (preview) | record Point(int x, int y) {} | N/A |
| Sealed classes (preview) | sealed interface Shape permits Circle, Rectangle {} | N/A |
| Enhanced enums | enum Color { RED(“#FF0000”), GREEN(“#00FF00”), BLUE(“#0000FF”) } | enum Color { RED(“#FF0000”), GREEN(“#00FF00”), BLUE(“#0000FF”); private String code; Color(String code) { this.code = code; } } |

Java 18 (released in March 2022)

| **Feature** | **Java 18** | **Prior to Java 18** |
| --- | --- | --- |
| Records | record Point(int x, int y) {} | N/A |
|  | Point p = new Point(1, 2); | N/A |
| Sealed Types | sealed interface Shape permits Circle, Square {} | N/A |
|  | final class Circle implements Shape {} | N/A |
|  | final class Square implements Shape {} | N/A |
| Pattern Matching for instanceof (Preview) | if (obj instanceof String s) { | N/A |
|  | // s has type String | N/A |
|  | } | N/A |
| Foreign-Memory Access (Incubating) | try (MemorySegment segment = MemorySegment.allocateNative(100)) { | N/A |
|  | // Use the memory segment | N/A |
|  | } | N/A |

Java 19 (released in September 2022)

* Records:
  + In Java 19, records have been further improved with the ability to specify custom “deep copy” methods, which are used to create a new instance of the record with copies of its components.

Copy coderecord Point(double x, double y) { public Point { if (x < 0 || y < 0) { throw new IllegalArgumentException("Coordinates must be non-negative"); } } // Custom deep copy method public Point copy() { return new Point(x, y); } }

* Enhanced Enums:
  + Java 19 introduces several new features for enums, including the ability to use the record keyword to define an enum with record components, and the ability to use the sealed keyword to restrict the inheritance of an enum.

Copy codesealed interface Shape permits Circle, Rectangle { } record Circle(double radius) implements Shape { } record Rectangle(double width, double height) implements Shape { }