✅Java 11 Features (Released: September 2018)

Java 11 is a Long-Term Support (LTS) version and a major upgrade from Java 8. It introduced multiple productivity and performance-focused enhancements.

---

1. HttpClient API (Standardized)

A modern HTTP client that supports synchronous and asynchronous calls. It replaces the legacy `HttpURLConnection`.

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create("https://example.com"))
    .build();

HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString());
System.out.println(response.body());
```

Supports asynchronous calls using `CompletableFuture` as well.

---

2. New String Methods

Method	Description
`isBlank()`	Returns true if the string is empty or contains only whitespace.
`lines()`	Returns a Stream of lines from the string.
`strip()`	Removes leading and trailing whitespace (Unicode-aware).
`stripLeading()`	Removes only leading whitespace (Unicode-aware).
`stripTrailing()`	Removes only trailing whitespace (Unicode-aware).
`repeat(n)`	Returns a new string with the original string repeated `n` times.

🔹 `strip()` vs `trim()`

Method	Unicode-aware	Removes Characters
`trim()`	❌	Removes characters with codepoint <= 32 (ASCII whitespace only)

Method	Unicode-aware	Removes Characters
`strip()`	✅	Removes all Unicode whitespace characters

🔹 `lines()` and Line Terminators

The `String.lines()` method splits a string into a Stream of lines using line terminators:

Recognized line terminators: - `\n` (Line Feed - LF) - `\r` (Carriage Return - CR) - `\r\n` (CRLF)

☸️**Note:** `lines()` only works with the above terminators. You **cannot** specify custom line delimiters (like `;` or `|`) — for that, use `split()` instead.

```
String text = "Line1\nLine2\rLine3\r\nLine4";
text.lines().forEach(System.out::println);
```

---

3. Local Variable Syntax for Lambda Parameters

You can now use `var` in lambda expressions:

```
List<String> list = List.of("Java", "Python");
list.forEach((var lang) -> System.out.println(lang));
```

🔹 Why use `var` in lambdas?

- Enables use of annotations on lambda parameters:

  ```
  list.forEach((@Nonnull var lang) -> System.out.println(lang));
  ```

- Improves consistency with local variable declarations using `var`.
- Useful when multiple parameters have the same annotation or need type inference.

Note: All parameters must use `var` if one does.

---

4. Collection API Enhancements

Java 11 (from Java 9) introduced immutable collection factory methods:

```
List<String> fruits = List.of("Apple", "Banana");
Set<Integer> nums = Set.of(1, 2, 3);
Map<String, String> map = Map.of("key", "value");
```

---

5. File API Enhancements

Added convenience methods:

```
Path path = Paths.get("sample.txt");
Files.writeString(path, "Hello Java 11");
String content = Files.readString(path);
System.out.println(content);
```

---

6. Enhancements to Optional Class

Java 11 introduced: - `Optional.isEmpty()` – returns true if the Optional is empty.

```
Optional<String> name = Optional.empty();
System.out.println(name.isEmpty()); // true
```

This complements `isPresent()` and improves code readability in some cases.