In **Java 8**, the forEach() method was introduced as part of the **Iterable interface** and also for **Streams**, enabling **clean, declarative, and functional-style iteration** over collections.

**🔹 What is forEach()?**

forEach() is used to **iterate over elements** of a collection or stream and **perform an action** on each element.

**🔰 Syntax**

**From Iterable:**

void forEach(Consumer<? super T> action);

**Parameters:**

* action: A lambda expression or method reference implementing the Consumer<T> interface.
* Consumer<T> → Functional Interface → void accept(T t)

**✅ Example with List**

List<String> names = Arrays.asList("Alice", "Bob", "Charlie");

// Using lambda

names.forEach(name -> System.out.println(name));

// Using method reference

names.forEach(System.out::println);

**✅ Example with Stream**

List<Integer> nums = Arrays.asList(1, 2, 3, 4);

nums.stream()

.filter(n -> n % 2 == 0)

.forEach(System.out::println); // Prints: 2 4

**🧠 Differences: forEach() on Collection vs Stream**

| **Feature** | **Iterable.forEach()** | **Stream.forEach()** |
| --- | --- | --- |
| Introduced in | Java 8 | Java 8 |
| Source | Collection (List, Set, etc.) | Stream pipeline |
| Supports lazy ops? | ❌ No | ✅ Yes (via stream pipeline) |
| Parallel support? | ❌ No | ✅ Use .parallelStream() |
| Order guaranteed? | ✅ Yes (List), ❌ (Set/Map) | Depends on stream source |

**✅ Parallel Processing with forEach()**

List<Integer> nums = Arrays.asList(1, 2, 3, 4, 5);

nums.parallelStream()

.forEach(n -> System.out.println(Thread.currentThread().getName() + " : " + n));

🔸 This processes elements in **parallel threads** (not guaranteed order).

**🚫 Caution: Do NOT modify collections inside forEach()**

List<String> list = new ArrayList<>(Arrays.asList("A", "B", "C"));

list.forEach(item -> {

if (item.equals("B"))

list.remove(item); // 🚫 ConcurrentModificationException

});

✅ Instead, use an iterator or removeIf:

list.removeIf(item -> item.equals("B"));

**🔁 Comparison: forEach() vs Traditional Loop**

| **Aspect** | **forEach()** | **for-loop / Iterator** |
| --- | --- | --- |
| Syntax | Clean, concise | Verbose |
| Functional style | ✅ Yes | ❌ No |
| Parallel support | ✅ with Streams | ❌ No |
| Break/Continue | ❌ Not allowed | ✅ Allowed |
| Mutating list | ❌ Unsafe | ✅ Safer with Iterator |

**💡 Real-World Example**

List<Employee> employees = Arrays.asList(

new Employee("John", 40000),

new Employee("Jane", 50000)

);

// Give 10% hike to all

employees.forEach(e -> e.setSalary(e.getSalary() \* 1.1));

**🧪 Custom forEach in Map**

Map<String, Integer> map = new HashMap<>();

map.put("A", 1);

map.put("B", 2);

map.forEach((key, value) -> System.out.println(key + " => " + value));

**📌 Summary**

| **Feature** | **Description** |
| --- | --- |
| Type | Terminal operation (on streams) |
| Input | Consumer<T> lambda or method reference |
| Used for | Iterating with side effects (e.g., print, mutate) |
| Supports parallel? | ✅ with .parallelStream() |
| Mutation-safe? | ❌ Use with caution on mutable collections |