Data Structures

Iterators

In Java, `Iterator` is an interface within the `java.util` package used to traverse the elements of a collection sequentially. It allows you to access and potentially remove elements from a collection without needing to know the collection's underlying implementation details.

Key methods of the Iterator interface:

- `boolean hasNext()`: This method returns `true` if there are more elements in the iteration, and `false` otherwise.
- `E next()`: This method returns the next element in the iteration and advances the iterator's internal pointer. It throws a `NoSuchElementException` if there are no more elements.
- `void remove()`: This optional method removes the last element returned by `next()` from the underlying collection. It can only be called once per call to `next()`, and attempting to call it at other times or multiple times after a single `next()` call will result in an `IllegalStateException`.

Iterator and the enhanced for-each loop:

The enhanced for-each loop (introduced in Java 5), which can be used for simple iteration without element removal, uses an Iterator internally. For example:

```
List<String> names = new ArrayList<>();
// Populate list
for (String name : names)
{
    System.out.println(name);
}
```

How to use an Iterator explicitly:

- Obtain an Iterator: Call the `iterator()` method on a Collection object (e.g., `ArrayList`, `HashSet`, `LinkedList`) to get an `Iterator` object for that collection.

  ```
  Iterator<String> it = names.iterator();
  ```

- Iterate through elements: Use a `while` loop with `hasNext()` and `next()` to process each element.

  ```
  while (it.hasNext())
  {
      String name = it.next();
      System.out.println(name);
  }
  ```

- Remove elements (optional): If needed, use the `remove()` method of the iterator to safely remove elements during iteration.

```
while (it.hasNext())
{
    String name = it.next();
    if (name.equals("John"))
    {
        it.remove(); // Removes "John" from the 'names' list
    }
}
```

Exceptions:

There are several exceptions that can be thrown by the `remove` method or other code when an `Iterator` is active.

- `UnsupportedOperationException`: If the `remove` operation is not supported by this iterator
- `IllegalStateException`: If the `next` method has not yet been called or the `remove` method has already been called after the last call to the `next` method.
- `ConcurrentModificationException`: Modifying the underlying collection directly (e.g., using `add()` or `remove()` methods of the collection itself) while an Iterator is active will lead to a `ConcurrentModificationException` when `next()` is called. Use the Iterator's `remove()` method for safe modification during iteration.

Additional Functionality:

- `ListIterator`: For `List` implementations, a more specialized `ListIterator` is available, which provides additional functionalities like bidirectional traversal (`hasPrevious()`, `previous()`) and element modification (`set()`, `add()`).
- `forEachRemaining()`: Since Java 8, the `Iterator` interface includes a `forEachRemaining()` method, allowing you to process remaining elements using a lambda expression.

From AI Overview after googling "Java iterators"; minor editing by Alyce Brady, 2 October, 2025.