UNIT IV – ITERATORS AND ENHANCED FOR LOOP

Enhanced for loop

- ✓ The enhanced for loop, also known as the "for-each" loop, was introduced in Java 5 to simplify iterating over arrays and collections.
- ✓ It abstracts away the complexity of managing an index or an iterator explicitly.
- ✓ The Java compiler automatically converts an enhanced for loop into standard for loop or Iterator calls behind the scenes.

Key characteristics of the enhanced for loop:

- ✓ More concise and readable than a standard for loop, especially for simple traversals.
- ✓ Reduces errors associated with manual index management, such as ArrayIndexOutOfBoundsException.
- ✓ Provides read-only access to the elements during iteration; you cannot modify the collection by adding or removing elements.
- ✓ Works with any class that implements the java.lang.Iterable interface.

Syntax of the enhanced for loop:

```
for (Type variable : collectionOrArray)
{
    // code to execute for each element
}
```

Example using an enhanced for loop

```
public class EnhancedForLoopArrayExample
{
    public static void main(String[] args)
    {
        int[] numbers = {12, 23, 44, 56, 78};
        System.out.println("Printing array elements using enhanced for loop:");
        for (int number : numbers)
        {
            System.out.println(number);
        }
    }
}
```

```
Printing array elements using enhanced for loop:
12
23
44
56
78
```

The loop iterates through the numbers array, assigning each element to the number variable in turn.

Example with an ArrayList

The enhanced for loop is also compatible with collections like

```
ArrayList. import java.util.ArrayList;
import java.util.List;
public class EnhancedForLoopListExample
{
    public static void main(String[] args)
    {
        List<String> fruits = new
        ArrayList<>(); fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Orange");
        System.out.println("Printing list elements using enhanced for
        loop:"); for (String fruit : fruits)
         {
            System.out.println(fruit);
        }
    }
}
```

OUTPUT:

```
Printing list elements using enhanced for loop:
 Apple
 Banana
 Orange
```

When to use the enhanced for loop:

Use the enhanced for loop for iterating through all elements of an array or collection in a forward, sequential manner, particularly when you don't need to modify elements or access their index. It enhances code readability and conciseness.

Limitations of the enhanced for loop:

Avoid using the enhanced for loop when you need to modify the collection or array, access element indices, iterate over only a portion of the data structure, or iterate in reverse order. In these cases, a traditional for loop is more appropriate.

ITERATORS

- An Iterator in Java is an interface that provides a standard way to traverse the elements of any collection, like ArrayList or HashSet, one by one.
- Iterators are a core part of the Java Collections Framework, offering a universal method for looping through a collection without needing to know its specific underlying implementation.

Key characteristics of an Iterator:

- Allows sequential access to elements without exposing the underlying collection structure.
- Supports both reading and removing elements from the collection while iterating.
- Works for any class that implements the java.util.Collection interface.
- Supports only forward-direction traversal (use a ListIterator for bi-directional traversal).

Key methods of the Iterator interface:

The Iterator<E> interface, found in the java.util package, includes three fundamental methods:

- **boolean hasNext():** Returns true if there are more elements to iterate through, and false otherwise.
- **next():** Returns the next element in the collection. You must first check with **hasNext()** to ensure another element exists, or this method will throw a noSuchElementException.
- **void remove():** Removes the last element that was returned by next(). This operation is optional and can only be called once per call to next().

Example of using an Iterator:

This example demonstrates how to use an Iterator to traverse and safely remove elements from an ArrayList while looping.

```
import java.util.ArrayList; import java.util.Iterator; public class IteratorExample
{
    public static void main(String[] args)
   {
        // Create an ArrayList of strings
        ArrayList<String> fruits = new ArrayList<>(); fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Orange");
        fruits.add("Grape");
        // Get the iterator for the ArrayList
        Iterator<String> iterator = fruits.iterator();
        System.out.println("Original list: " + fruits);
        // Iterate through the collection and remove an element while
        (iterator.hasNext())
       {
            String currentFruit = iterator.next();
            System.out.println("Processing: " + currentFruit);
            // Check for and remove a specific element if
            (currentFruit.equals("Orange"))
            {
                System.out.println("Removing 'Orange'");
                iterator.remove(); // Safely removes the element
            }
        }
        System.out.println("List after removal: " + fruits);
    }
}
```

OUTPUT:

```
Original list: [Apple, Banana, Orange, Grape]
Processing: Apple
Processing: Banana
Processing: Orange
Removing 'Orange'
Processing: Grape
List after removal: [Apple, Banana, Grape]
```

Advantages of using an Iterator:

- **Decouples collection and traversal logic:** Using an Iterator separates the logic of traversing a collection from the collection's internal structure. This allows you to iterate over different types of collections (e.g., ArrayList, HashSet) with the same code.
- **Safe element removal:** Iterators provide a safe way to remove elements from a collection during a traversal. Direct modification of the collection while iterating with a simple for loop would cause a ConcurrentModificationException.
- **Universal usage:** The Iterator interface is implemented by all classes in the Java Collections Framework, making it a standard and versatile tool.

Comparison with for-each loop

- The enhanced for loop (or for-each loop) is a syntactic shortcut that internally uses an iterator.
- While it is more concise for simple traversals, it lacks the ability to remove elements during the loop.

for-each loop:

```
// This will throw a ConcurrentModificationException
    ArrayList<String> fruits = new ArrayList<>();
    fruits.add("Apple");
    for (String fruit : fruits)
    {
      if (fruit.equals("Apple"))
      {
        fruits.remove(fruit);
      }
    }
```

Iterator loop: This is the correct and safe way to remove

```
elements ArrayList<String> fruits = new ArrayList<>();
fruits.add("Apple");
Iterator<String> iterator =
fruits.iterator(); while (iterator.hasNext())
{
    String fruit =
    iterator.next(); if
    (fruit.equals("Apple"))
    {
        iterator.remove();
    }
}
```