Enumeration in Java Collection Framework

What?

* It is an Interface.
* Part of java.util
* It provides methods to enumerate(traverse) elements one by one in collection
* Primarily used for legacy classes (stack, vector and hashtable)

Key Points:

Methods:

* Boolean hasMoreElements() : Returns true if there are more elements to traverse.
* nextElement() : Returns the next element in enumeration.

Limitations:

* It only support the forward traversal
* It is read-only(we cannot remove element during iteration)
* package Enumeration;  
    
  import java.util.Enumeration;  
  import java.util.Vector;  
    
  public class EnumerationExample {  
   public static void main(String[] args) {  
   // Creating vector and adding elements  
   Vector<String> vector = new Vector<>();  
   vector.add("Java");  
   vector.add("Python");  
   vector.add("C++");  
   vector.add("JS");  
    
   //Get Enumeration  
   Enumeration<String> enumeration = vector.elements();  
    
   //Traverse the elements  
   System.*out*.println("Elements in vector :");  
   while(enumeration.hasMoreElements()){  
   System.*out*.println(enumeration.nextElement());  
   }  
   }  
  }

vector.elements(): Returns an Enumeration of elements in the vector

Assignement:

Traverse a Stack using Enumeration

Traverse a Hashtable using Enumeration(keys() & elements())

Iterators:

Iterators are used to traverse elements of a collection sequentially.

They provide a way to access elements one by one

Key Points:

1. It is Interface.

* Part of java.util
* Supports sequential traversal

1. Methods

* Boolean hasNext() – Checks if the collection has more elements to traverse
* E next() – Returns the next element in the collection
* void remove() – Removes the last element returned by the iterator

Why Iterators?

* Works with various data structures like ArrayList, HashSet, etc
* Ensures fail-fast behavior (throws ConcurrentModificationException if the collection is modified Structurally during iteration)

Types of Iterators:

* Iterator -> Allows read and remove operation during iteration
* ListIterator -> Extends Iterator, allows bidirectional traversal(Only for List collections like ArrayList or LinkedList)
* Enumeration -> Legacy Interface used with stack and vector

|  |  |  |
| --- | --- | --- |
| Feature | Iterator | Enumeration |
| Introduced in | Java 2 | Java 1 |
| Methods | hasNext(),next(),remove() | hasMoreElements(), nextElement() |
| Usage | All collections in the framework | Legacy classes(Vector, stack) |
| Modification | Can remove elements during iteration | No modification is allowed. |

package Iterator;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Iterator;  
import java.util.List;  
  
public class IteratorExample {  
 public static void main(String[] args) {  
 List<String> list = new ArrayList<>(Arrays.*asList*("Java","Python", "C++","CS"));  
  
 Iterator<String> iterator = list.iterator();  
  
 System.*out*.println("Iterating using iterator : ");  
 while(iterator.hasNext()){  
 String element = iterator.next();  
 System.*out*.println(element);  
 if(element.equals("Python")){  
 iterator.remove(); // Removes "Python" from list  
 }  
 }  
 System.*out*.println("Modified list :" + list);  
 }  
}

package Iterator;  
  
import java.util.\*;  
  
public class ListIteratorExample {  
 public static void main(String[] args) {  
 List<String> list = new ArrayList<>(Arrays.*asList*("Java","Python", "C++","CS"));  
 ListIterator<String> listIterator = list.listIterator();  
  
 System.*out*.println("Forward traversal : ");  
 while(listIterator.hasNext()){  
 System.*out*.println(listIterator.next());  
 }  
  
 System.*out*.println("\nBackward traversal :");  
 while(listIterator.hasPrevious()){  
 System.*out*.println(listIterator.previous());  
 }  
  
 }  
}