Application Program Development

Segment : Java Iterators

Mahboob Ali

Outcomes

- Understanding Iterators.
- Understanding Iterators vs Iterable.
- C++ STL iterator vs Java Iterator

Iterators

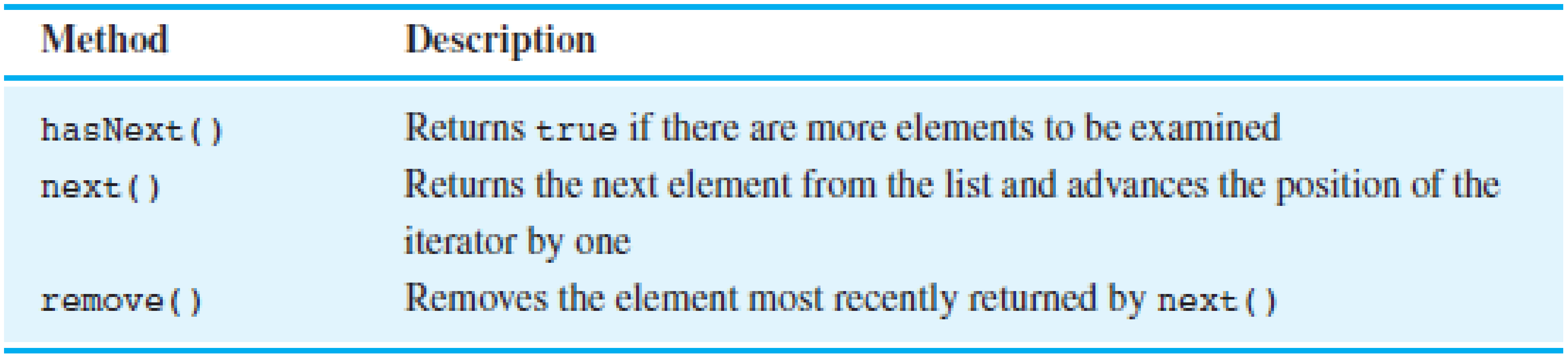
- **Iterator**: An Iterator is a construct/ object that is used to traverse or step through the collection.
- Java Iterator is a collection framework interface and is a part of the “java.util” package.
- Using Java Iterator, you can iterate through the collection of objects.
- **Iterator Types**
    - Enumerator
    - Iterator
    - ListIterator

Enumerators

- Java Iterator interface replaces the enumerator that was used earlier to step through some simple collections like vectors.

Iterator Interface

- The Iterator interface is available from the Java 1.2 collection framework onwards.
- It traverses the collection of objects one by one.
- Popularly known as "Universal Java Cursor" as it works with all collections.
- This interface supports 'read' and 'remove' operations i.e. you can remove an element during an iteration using the iterator.



Method	Description
`hasNext()`	Returns `true` if there are more elements to be examined
`next()`	Returns the next element from the list and advances the position of the iterator by one
`remove()`	Removes the element most recently returned by `next()`

The next() method

- **Prototype:** E next ()
- **Parameters:** no parameters
- **Return type:** E -> element
- **Description:** Returns the next element in the collection.
- If the iteration (collection) has no more elements, then it throws **NoSuchElementException**.

The hasNext() method

- **Prototype:** boolean hasNext()

- **Parameters:** NIL

- **Return type:** true => there are elements in the collection.
  False => no more elements

- **Description:** The function hasNext() checks if there are more elements in the collection that is being accessed using an iterator. If there are no more elements, then you don't call the next() method. In other words, this function can be used to decide if the next() method is to be called.

The remove() method

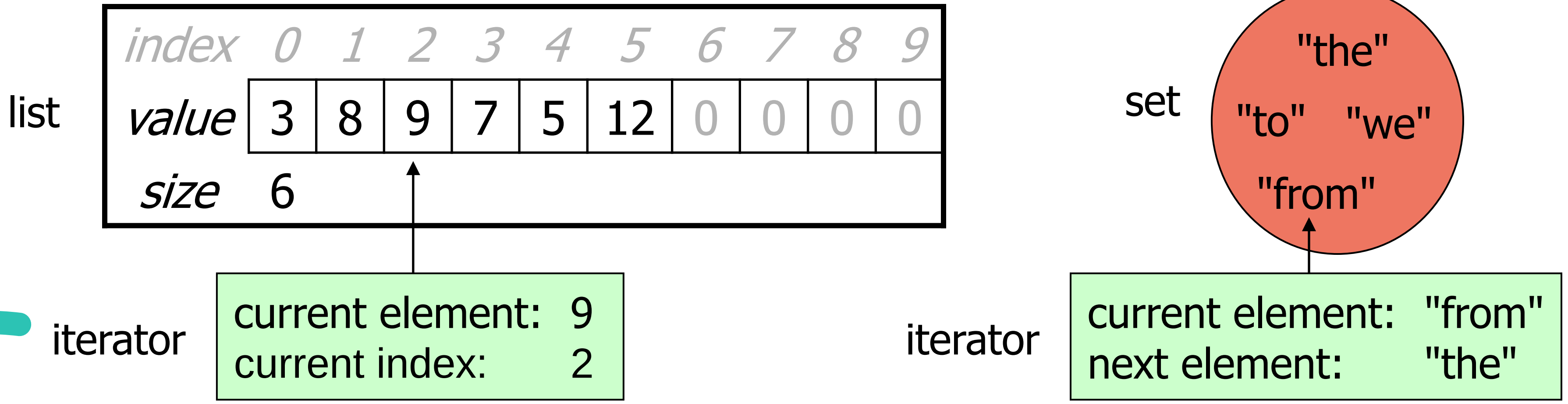
- **Prototype:** void remove()
- **Parameters:** NIL
- **Return type:** NIL

- **Description:** Removes the last element returned by the iterator iterating over the underlying collection. The remove () method can be called only once per next () call.

- If the iterator does not support remove operation, then it throws **UnSupportedOperationException**. It throws **IllegalStateException** if the next method is not yet called.

Rules

- Remembers a position, and lets you:
    - get the element at that position
    - advance to the next position
    - remove the element at that position

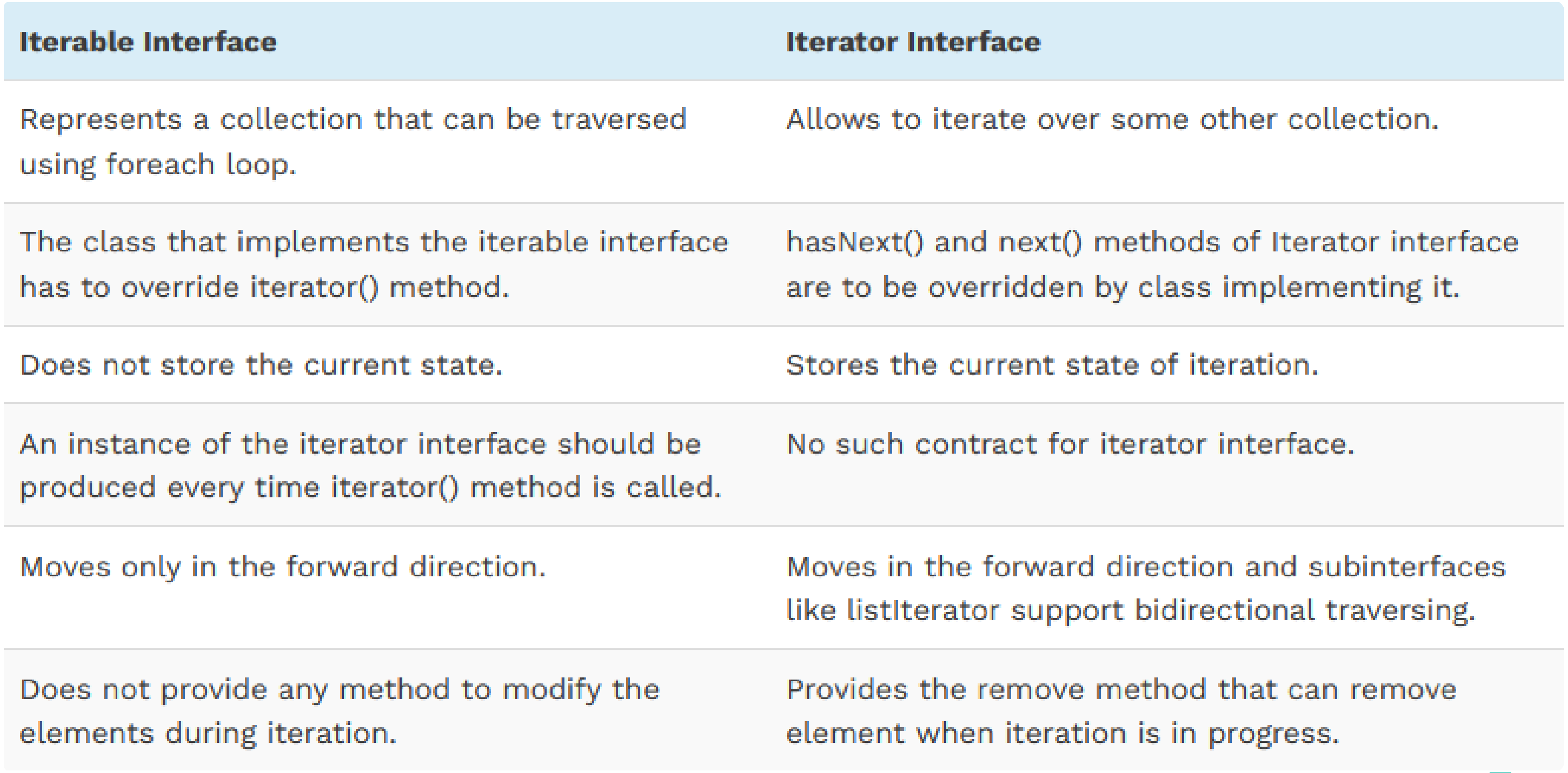


Iterator example

```
import java.util.*;

public class IteratorExample
{
  public static void main(String[] args)
  {
         List<String> flowers = new ArrayList<String>();
         flowers.add("Rose");
         flowers.add("Jasmine");
         flowers.add("sunflower");


         // Get Iterator
         Iterator<String>flowersIterator = flowers.iterator();

         System.out.println("Contents of ArrayList:");
         // Traverse elements using iterator
         while(flowersIterator.hasNext()){
              System.out.print(flowersIterator.next() + " ");
         }
  }
}
```

Limitations of Iterator Interface

- The operation to replace an element or add a new element cannot be performed with this Iterator.
- The iteration proceeds only in one direction i.e. the forward direction.
- Supports only sequential iteration.
- When large volumes of data are to be iterated, then the performance of the Iterator is affected.



Iterable Interface	Iterator Interface
Represents a collection that can be traversed using foreach loop.	Allows to iterate over some other collection.
The class that implements the iterable interface has to override iterator() method.	hasNext() and next() methods of Iterator interface are to be overridden by class implementing it.
Does not store the current state.	Stores the current state of iteration.
An instance of the iterator interface should be produced every time iterator() method is called.	No such contract for iterator interface.
Moves only in the forward direction.	Moves in the forward direction and subinterfaces like listIterator support bidirectional traversing.
Does not provide any method to modify the elements during iteration.	Provides the remove method that can remove element when iteration is in progress.

Iterator VS Iterable

- Though the interfaces Iterable and Iterator sound similar, they are completely different.
- A class that implements the Iterable interface acquires the ability to iterate over the class objects that use the iterator interface.

Java Iterators VS C++ Iterator

Java Iterator	C++ Iterator
Java Iterator is an interface, with different functions and has a concept of limit for the container being traversed.	In the C++ standard template library, iterators seem to represent a datatype or class the supports the operator++ and operator== but has no concept of a limit built in so comparison is required before advancing to the next item.
	The limit has to checked by the user comparing two iterators in the normal case the second iterator is the container end.
Java iterators correspond to C++' input iterators’, i.e., they are read-only iterators that can only be incremented one step at a time and can't go backwards	C++ iterators can go backwards as well, write, random etc.
Java iterators describe an intermixed position(state)-and-range(value)	C++ iterators separate the concepts of position and range. C++ iterators represent 'where am I now' separately from 'where can I go?'.
Java iterators can't be copied. You can't recover a previous position.	The common C++ iterators can.