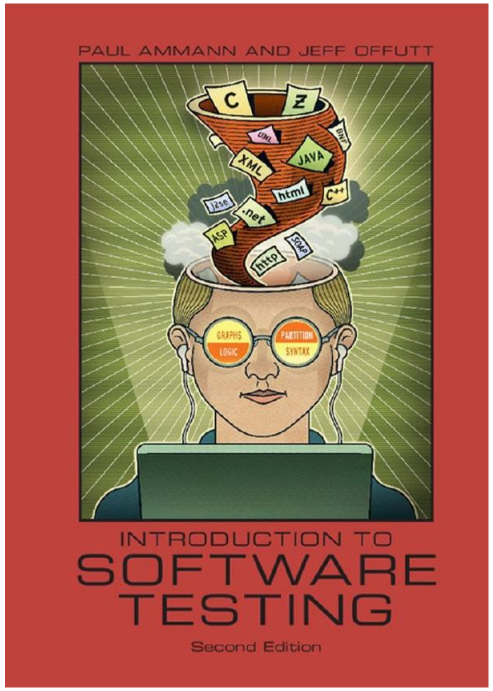
Exercise :
Testing the Iterator<List> class

Bibliografia

- **Paul Ammann and Jeff Offutt, Introduction to Software testing, 2nd Edition, Cambridge University Press**



Objective

- **The objective of this exercise are :**
  - To plan the testing of a class starting from the knowledge of its requirements and documentation
  - To exercise with Junit

Class under Testing

- **We want to test the Iterator class**
- **Iterator is not exactly a class: it is declared as an interface including a template**

```
Interface Iterator<E>
```

- **It can be used in the context of a collection providing an iterator to access sequentially to its items**

Example of use

```
 List<String> list;
 Iterator<String> itr;

list = new ArrayList<String>();
list.add ("cat");
list.add ("dog");
itr = list.iterator();
```

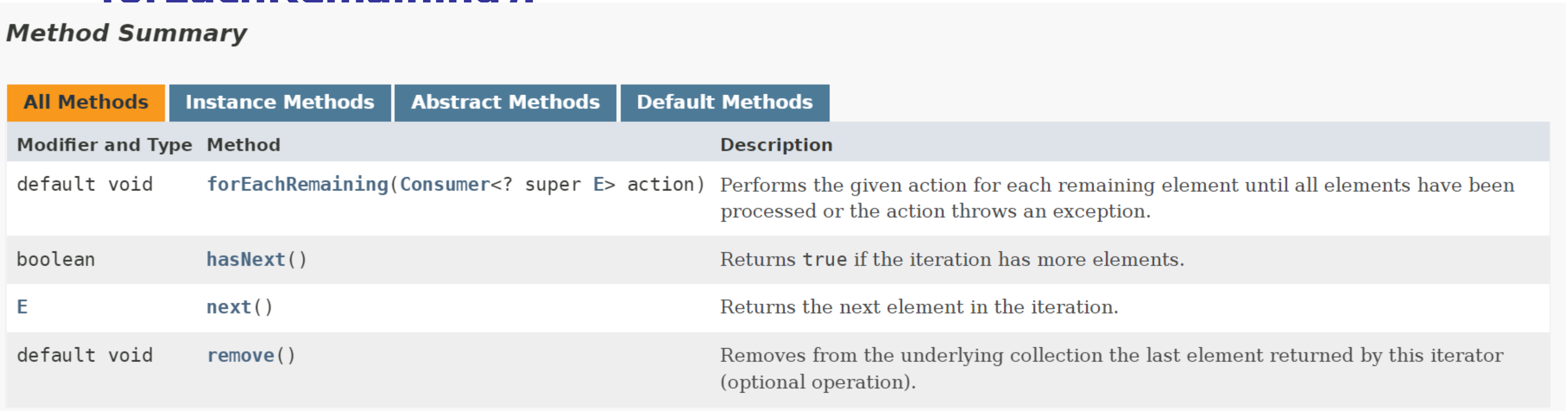
- **List provides an implementation of iterator**
  - We will test this implementation
  - Maybe, the test cases could be adapted to other implementations of iterator
- **On the itr object we can execute iterator methods**

Iterator documentation

The official documentation of Iterator is at:

https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/Iterator.html

In particular, the methods that we want to test are (excluding forEachRemaining):



Method Summary

All Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method	Description
`default void`	`forEachRemaining(Consumer<? super E> action)`	Performs the given action for each remaining element until all elements have been processed or the action throws an exception.
`boolean`	`hasNext()`	Returns `true` if the iteration has more elements.
`E`	`next()`	Returns the next element in the iteration.
`default void`	`remove()`	Removes from the underlying collection the last element returned by this iterator (optional operation).

Iterator Documentation

hasNext() – Returns true if there are more elements

- Exception: NullPointerException

E next() – Returns next element

- Exception: NoSuchElementException

void remove() – Removes the most recent element returned by the iterator

- Exception: Unsupported-OperationException
- Exception: IllegalStateException

parameters: state of the iterator

- iterator state changes with next(), and remove() calls
- modifying underlying collection also changes iterator state

hasNext

- **Execution scenarios**
    - There is another item
    - There are not more items
    - Iterator is null

hasNext

- **Execution scenarios**
  - There is another item
  - There are not more items
  - Iterator is null

ID	Precond	Input	ExpectedOutput	Output	PostCond	Result
1	List has two items (cat,dog) & an Iterator	hasNext	True	True	List and iterator null	OK
2	List has two items (cat,dog) & an Iterator	Next Next hasNext	False	False	List and iterator null	OK
3	List has two items (cat,dog) & an Iterator	Iterator null hasNext	NullPointerException	NullPointerException	List and iterator null	OK

Setup and Teardown

- First of all, declare the needed objects in the test class

```
private List<String> list; // test fixture
private Iterator<String> itr; // test fixture
```

- Then, implement setup (@BeforeEach) and teardown (@AfterEach) methods have to be implemented
  - We need different test classes if we have different setup and teardown methods

Excerpt of setup and tearDown

```
@BeforeEach
public void setUp() // set up test fixture
{
list = new ArrayList<String>();
…
assume…
}


@AfterEach
public void tearDown()
{
list=null;
…
assume…
}
```

Example of test method

```
// Test 1 of hasNext(): testHasNext_BaseCase()
  @Test
 public void testHasNext_BaseCase()    {
      assertTrue (itr.hasNext());
  }
```

next : suggested scenarios to be tested

- **Execution scenarios**
  - Read the first item
  - Read beyond the last item
  - Iterator is null

remove : suggested scenarios to be tested

- **Execution scenarios**
  - Remove the first item
  - Remove the last item
  - Remove without selecting an item
  - Remove an invalid iterator
  - Remove all items
  - Remove two times the same item

remove : exception scenarios

- **Exception scenarios**
  - An illegal state exception is raised if we try to remove without having selected an item
  - If the list is set as unmodifiable, an unsupported operation exception is raised when removing an item