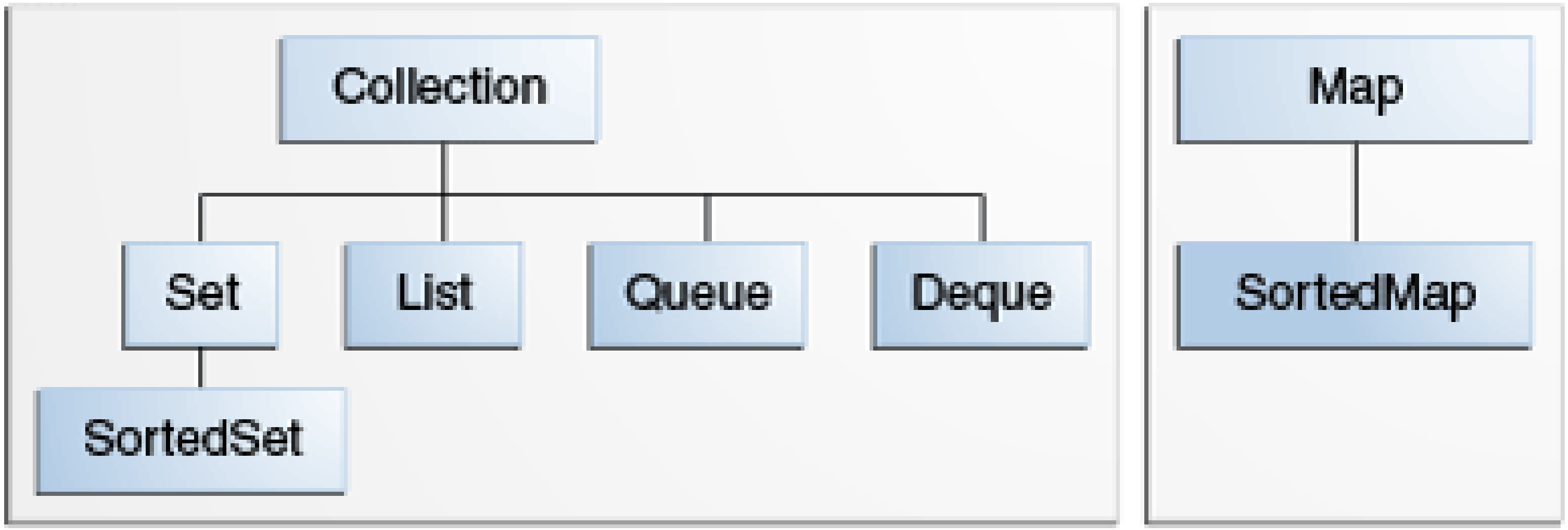
Structuri de date în Java

Java Collections Framework

Deque, Iterator

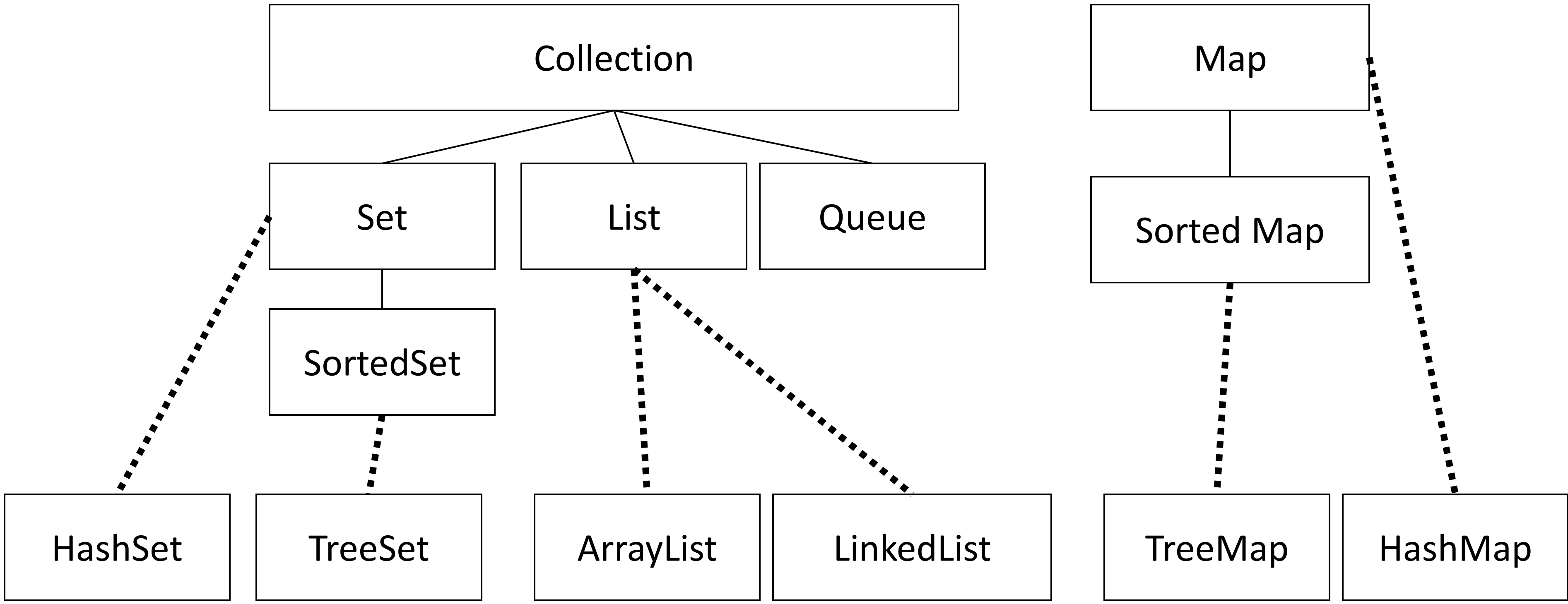
The core collection interfaces



Collection Interface

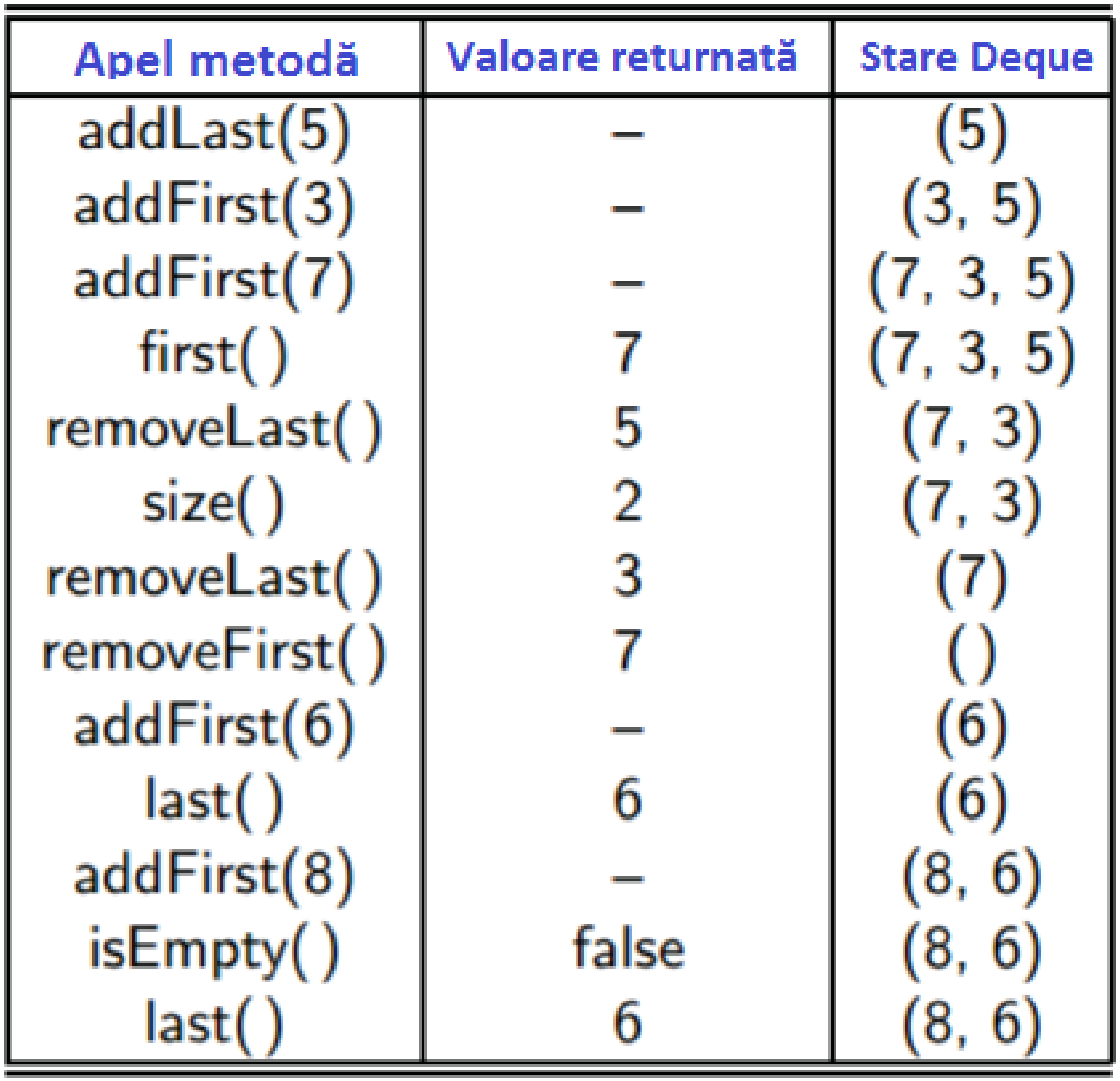
- Operatii de baza
  - int size();
  - boolean isEmpty();
  - boolean contains(Object element);
  - boolean add(E element);
  - boolean remove(Object element);
  - Iterator iterator();
- Operatii cu colectii
  - boolean containsAll(Collection<?> c);
  - boolean addAll(Collection<? extends E> c);
  - boolean removeAll(Collection<?> c);
  - boolean retainAll(Collection<?> c);
  - void clear();
- Operatii cu vectori
  - Object[] toArray(); <T> T[] toArray(T[] a); }

General Purpose Implementations



```
List<String> list1 = new ArrayList<String>(c);
List<String> list2 = new LinkedList<String>(c);
```

Double Ended Queue (Deque)

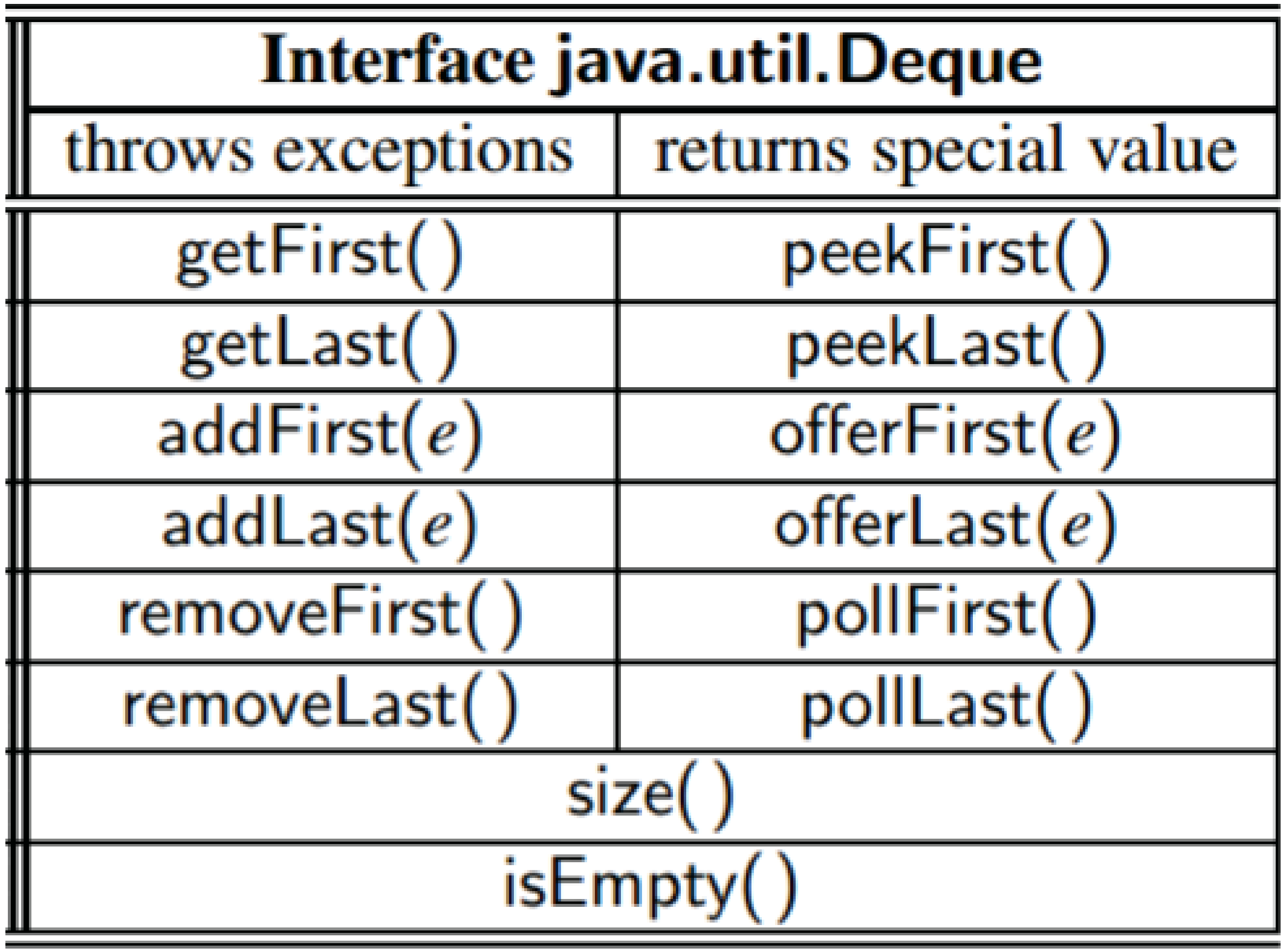


Apel metodă	Valoare returnată	Stare Deque
addLast(5)	–	(5)
addFirst(3)	–	(3, 5)
addFirst(7)	–	(7, 3, 5)
first()	7	(7, 3, 5)
removeLast()	5	(7, 3)
size()	2	(7, 3)
removeLast()	3	(7)
removeFirst()	7	()
addFirst(6)	–	(6)
last()	6	(6)
addFirst(8)	–	(8, 6)
isEmpty()	false	(8, 6)
last()	6	(8, 6)

Double Ended Queue (Deque)

```
public interface Deque<E> {    // o descriere de principiu
    int size( );
    boolean isEmpty( );
    E getFirst( );
    E getLast( );
    void addFirst(E e);
    void addLast(E e);
    E removeFirst( );
    E removeLast( );
}
```

Double Ended Queue (Deque)

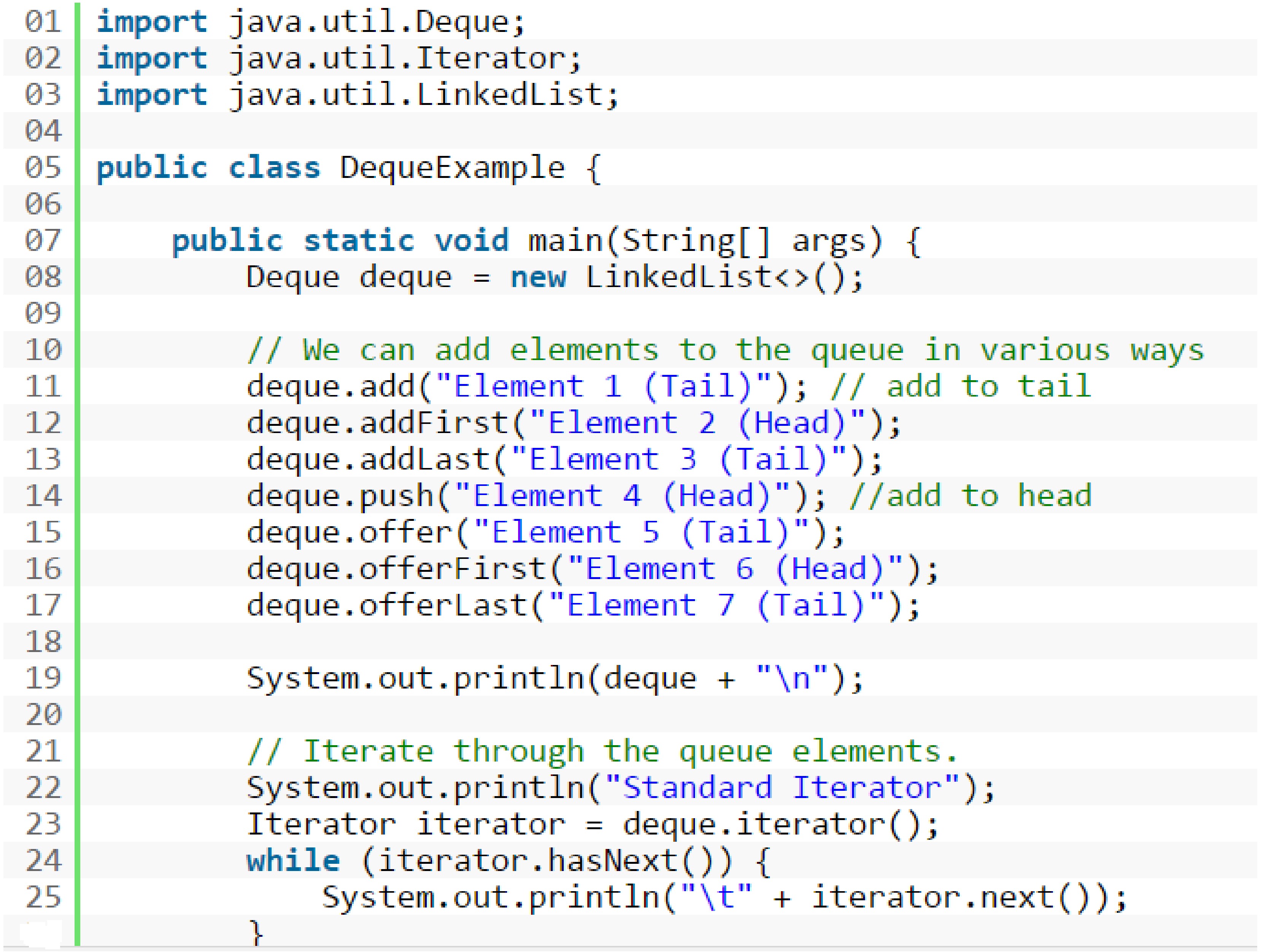


Interface java.util.Deque	
throws exceptions	returns special value
getFirst()	peekFirst()
getLast()	peekLast()
addFirst(e)	offerFirst(e)
addLast(e)	offerLast(e)
removeFirst()	pollFirst()
removeLast()	pollLast()
size()	
isEmpty()	

NoSuchElementExcepton (getFirst(), getLast(), removeFirst(), removeLast())
null (peekFirst(), peekLast(), pollFirst(), pollLast())

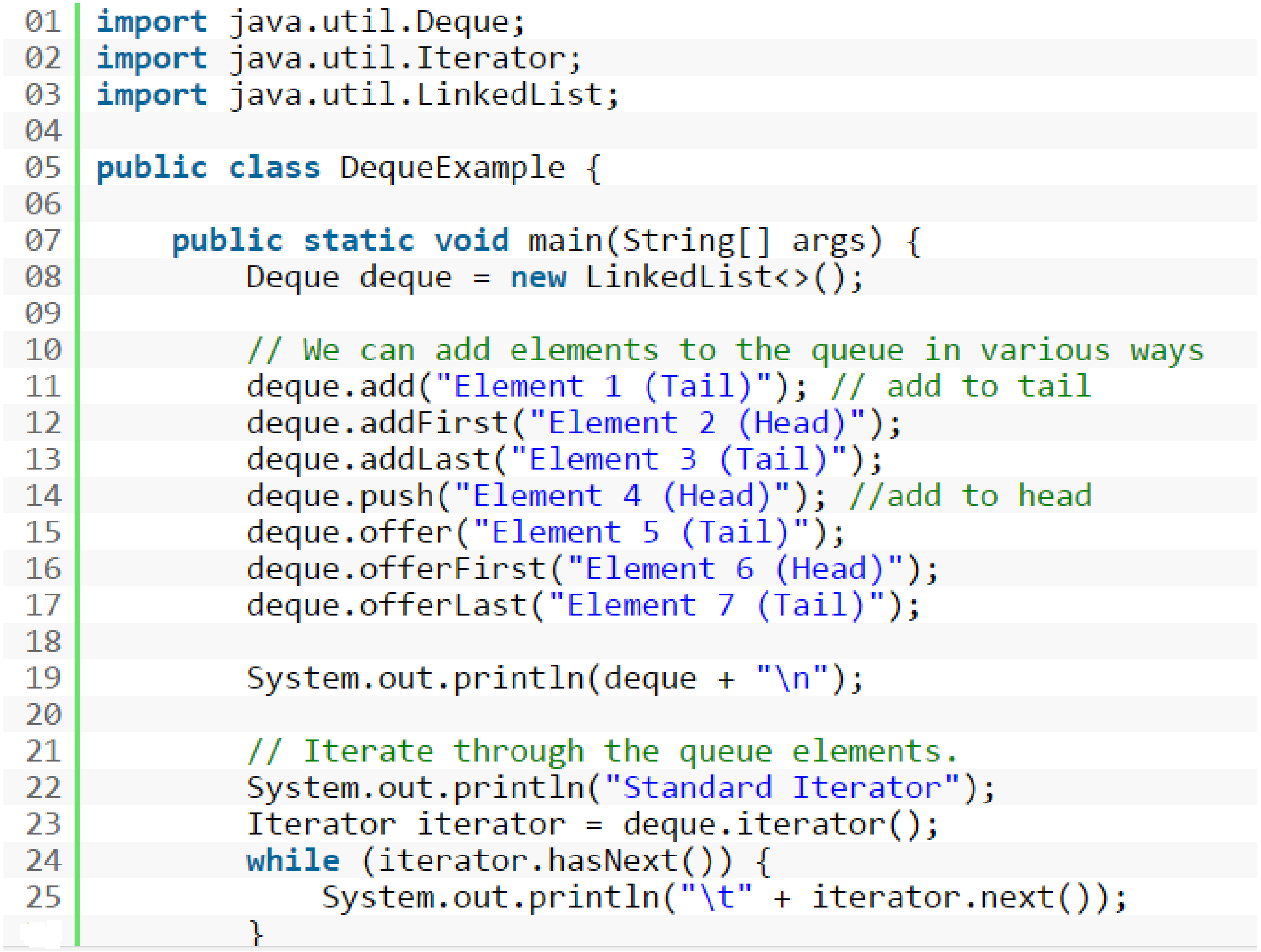
Implementări: java.util.ArrayDeque java.util.LinkedList

```
Deque dequeA = new LinkedList();

dequeA.add("e0");  // la urma;  dequeA = (e0)
dequeA.addLast("e1");   //      dequeA = (e0,e1)
dequeA.addFirst("e2");  //      dequeA = (e2 e0 e1)

Iterator iterator = dequeA.iterator();
while(iterator.hasNext(){
  String element = (String) iterator.next();
}

for(Object object : dequeA) {
    String element = (String) object;
}
Object p = dequeA.removeFirst(); // p=e2,  dequeA = ( e0 e1 )
Object u = dequeA.removeLast(); // u=e1 ,  dequeA = ( e0 )
```



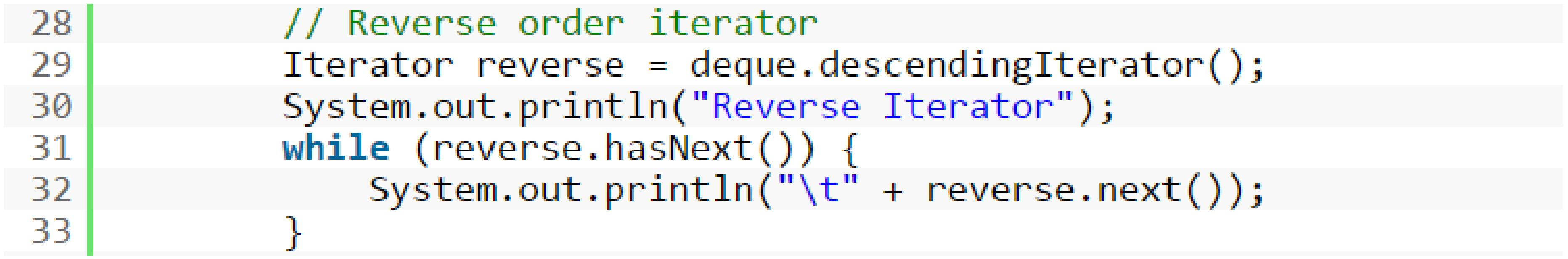
```java
import java.util.Deque;
import java.util.Iterator;
import java.util.LinkedList;

public class DequeExample {

    public static void main(String[] args) {
        Deque deque = new LinkedList<>();

        // We can add elements to the queue in various ways
        deque.add("Element 1 (Tail)"); // add to tail
        deque.addFirst("Element 2 (Head)");
        deque.addLast("Element 3 (Tail)");
        deque.push("Element 4 (Head)"); //add to head
        deque.offer("Element 5 (Tail)");
        deque.offerFirst("Element 6 (Head)");
        deque.offerLast("Element 7 (Tail)");

        System.out.println(deque + "\n");

        // Iterate through the queue elements.
        System.out.println("Standard Iterator");
        Iterator iterator = deque.iterator();
        while (iterator.hasNext()) {
            System.out.println("\t" + iterator.next());
        }
```

Ce afiseaza **System.out.println(deque +”\n”);** ?



```java
import java.util.Deque;
import java.util.Iterator;
import java.util.LinkedList;

public class DequeExample {

    public static void main(String[] args) {
        Deque deque = new LinkedList<>();

        // We can add elements to the queue in various ways
        deque.add("Element 1 (Tail)"); // add to tail
        deque.addFirst("Element 2 (Head)");
        deque.addLast("Element 3 (Tail)");
        deque.push("Element 4 (Head)"); //add to head
        deque.offer("Element 5 (Tail)");
        deque.offerFirst("Element 6 (Head)");
        deque.offerLast("Element 7 (Tail)");

        System.out.println(deque + "\n");

        // Iterate through the queue elements.
        System.out.println("Standard Iterator");
        Iterator iterator = deque.iterator();
        while (iterator.hasNext()) {
            System.out.println("\t" + iterator.next());
        }
```

[Element 6 (Head), Element 4 (Head), Element 2 (Head), Element 1 (Tail), Element 3 (Tail), Element 5 (Tail), Element 7 (Tail)]

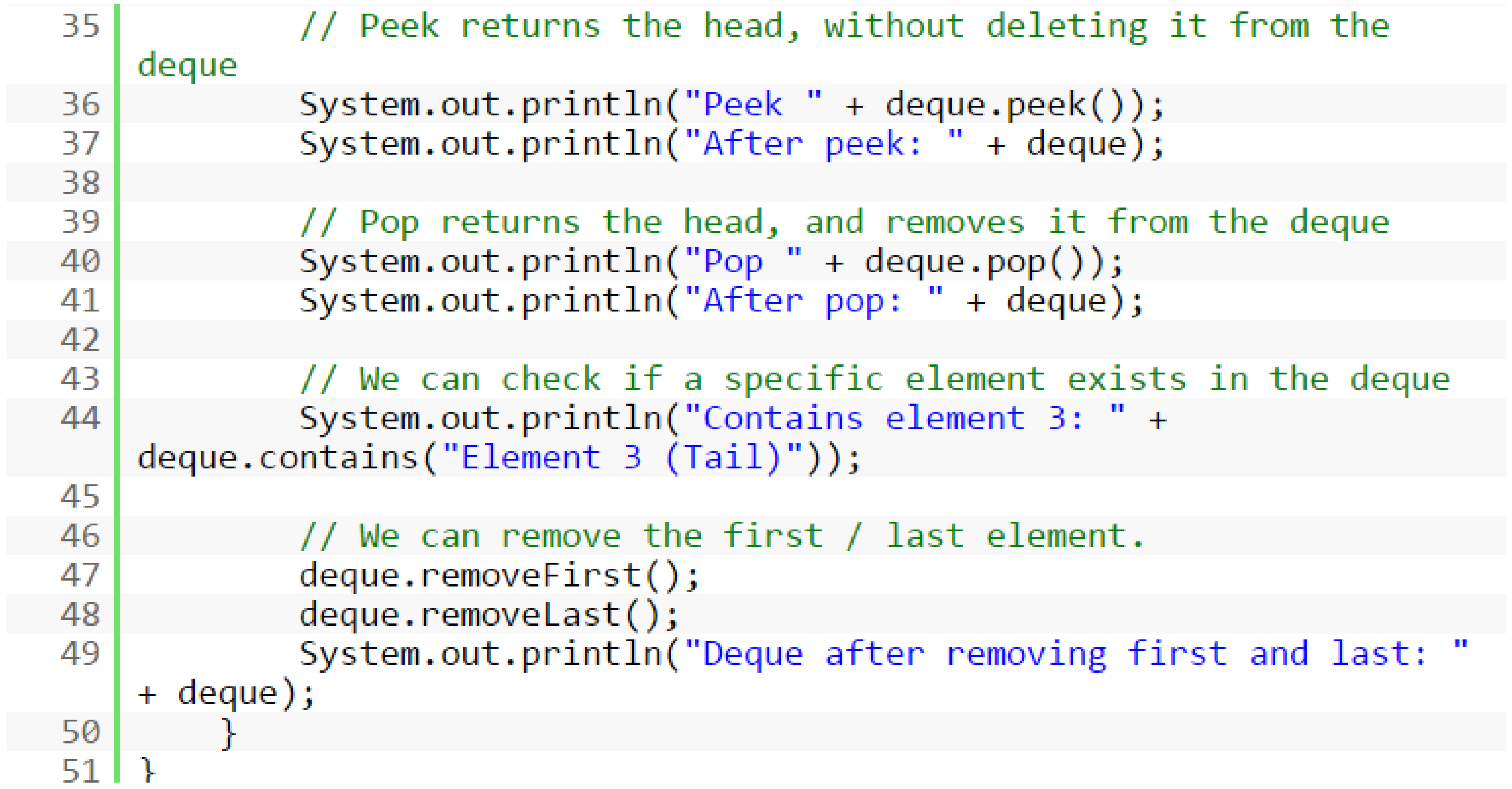


```
        // Reverse order iterator
        Iterator reverse = deque.descendingIterator();
        System.out.println("Reverse Iterator");
        while (reverse.hasNext()) {
            System.out.println("\t" + reverse.next());
        }
```

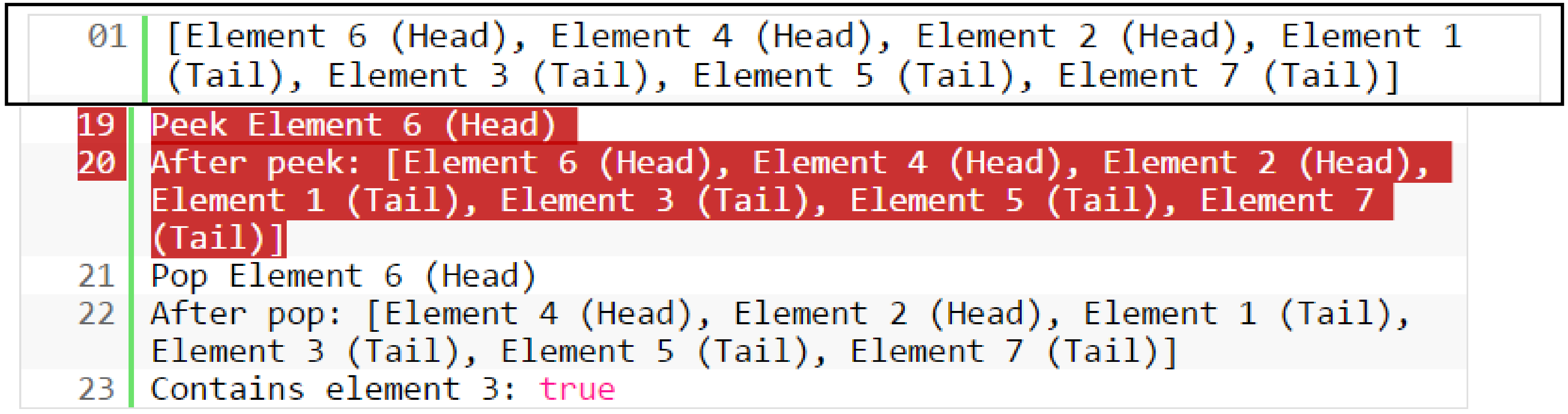
Ce afiseaza Iterator-ul standard si ‘descending’ ?

```
        // Reverse order iterator
        Iterator reverse = deque.descendingIterator();
        System.out.println("Reverse Iterator");
        while (reverse.hasNext()) {
            System.out.println("\t" + reverse.next());
        }
```

```
[Element 6 (Head), Element 4 (Head), Element 2 (Head), Element 1
(Tail), Element 3 (Tail), Element 5 (Tail), Element 7 (Tail)]

Standard Iterator
	Element 6 (Head)
	Element 4 (Head)
	Element 2 (Head)
	Element 1 (Tail)
	Element 3 (Tail)
	Element 5 (Tail)
	Element 7 (Tail)
Reverse Iterator
	Element 7 (Tail)
	Element 5 (Tail)
	Element 3 (Tail)
	Element 1 (Tail)
	Element 2 (Head)
	Element 4 (Head)
	Element 6 (Head)
```



```java
        // Peek returns the head, without deleting it from the deque
        System.out.println("Peek " + deque.peek());
        System.out.println("After peek: " + deque);

        // Pop returns the head, and removes it from the deque
        System.out.println("Pop " + deque.pop());
        System.out.println("After pop: " + deque);

        // We can check if a specific element exists in the deque
        System.out.println("Contains element 3: " + deque.contains("Element 3 (Tail)"));

        // We can remove the first / last element.
        deque.removeFirst();
        deque.removeLast();
        System.out.println("Deque after removing first and last: " + deque);
    }
}
```



```
[Element 6 (Head), Element 4 (Head), Element 2 (Head), Element 1 (Tail), Element 3 (Tail), Element 5 (Tail), Element 7 (Tail)]
Peek Element 6 (Head)
After peek: [Element 6 (Head), Element 4 (Head), Element 2 (Head), Element 1 (Tail), Element 3 (Tail), Element 5 (Tail), Element 7 (Tail)]
Pop Element 6 (Head)
After pop: [Element 4 (Head), Element 2 (Head), Element 1 (Tail), Element 3 (Tail), Element 5 (Tail), Element 7 (Tail)]
Contains element 3: true
```

Bibliografie

https://examples.javacodegeeks.com/core-java/util/deque-util/java-util-deque-example/
http://tutorials.jenkov.com/java-collections/deque.html#implementations