# Folder src/util

**3 printable files**

## (file list disabled)

**src/util/It.java**

```java
/**
 * Classe It représentant un itérateur générique pour une structure de données chaînée.
 * Permet de parcourir les éléments d'une liste chaînée de manière séquentielle.
 *
 * @param <T> le type des éléments à parcourir.
 * @Author : Maxime Lestiboudois
 * @Author : Nathan Parisod
 * @date : 27/11/2024
 */
package util;

import java.util.NoSuchElementException;

public class It<T> {
    /**
     * Le nœud courant de la liste chaînée.
     */
    Node<T> current;

    /**
     * Constructeur de l'itérateur.
     * Initialise l'itérateur à partir d'un nœud de départ.
     *
     * @param start le nœud de départ de l'itérateur.
     */
    public It(Node<T> start) {
        this.current = start;
    }

    /**
     * Vérifie si l'itérateur a un élément suivant.
     *
     * @return true si un élément suivant existe, false sinon.
     */
    public boolean hasNext() {
        return current != null;
    }

    /**
     * Renvoie l'élément suivant dans la liste chaînée et avance l'itérateur.
     *
     * @return la valeur de l'élément courant.
     * @throws NoSuchElementException si aucun élément suivant n'existe.
     */
    public T next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        T value = current.data;
        current = current.next;
        return value;
    }
}
```

**src/util/Node.java**

```java
/**
 * Classe Node représentant un nœud d'une liste chaînée.
 * Chaque nœud contient une donnée et une référence vers le nœud suivant dans la liste.
 *
 * @param <T> le type de la donnée contenue dans le nœud.
 * @Author : Maxime Lestiboudois
 * @Author : Nathan Parisod
 * @date : 27/11/2024
 */
package util;

class Node<T> {
    /**
     * La donnée stockée dans ce nœud.
     */
    T data;

    /**
     * La référence vers le nœud suivant dans la liste.
     */
    Node<T> next;

    /**
     * Constructeur de la classe Node.
     * Initialise le nœud avec une donnée et sans nœud suivant.
     *
     * @param data la donnée à stocker dans le nœud.
     */
    Node(T data) {
        this.data = data;
        this.next = null;
    }
}
```

**src/util/Stack.java**

```java
/**
 * Classe Stack représentant une pile générique basée sur une structure de liste chaînée.
 * Permet d'insérer, retirer et manipuler des éléments selon le principe LIFO (Last In, First Out).
 *
 * @param <T> le type des éléments stockés dans la pile.
 * @Author : Maxime Lestiboudois
 * @Author : Nathan Parisod
 * @date : 27/11/2024
 */
package util;

import java.util.NoSuchElementException;

public class Stack<T> {
    /**
     * Référence au sommet de la pile.
     */
    private Node<T> head;

    /**
     * Constructeur de la pile.
     * Initialise une pile vide.
     */
    public Stack() {
        this.head = null;
    }

    /**
     * Insère un nouvel élément au sommet de la pile.
```

```java
     *
     * @param value la valeur à insérer.
     */
    public void insert(T value) {
        Node<T> newNode = new Node<>(value);
        newNode.next = head;
        head = newNode;
    }

    /**
     * Vide la pile en supprimant tous ses éléments.
     */
    public void clear() {
        head = null;
    }

    /**
     * Retire et renvoie l'élément au sommet de la pile.
     *
     * @return la valeur de l'élément retiré.
     * @throws NoSuchElementException si la pile est vide.
     */
    public T pop() {
        if (isEmpty()) {
            throw new NoSuchElementException("Stack is empty");
        }
        T value = head.data;
        head = head.next;
        return value;
    }

    /**
     * Vérifie si la pile est vide.
     *
     * @return true si la pile est vide, false sinon.
     */
    public boolean isEmpty() {
        return head == null;
    }

    /**
     * Renvoie une représentation sous forme de chaîne de caractères des éléments de la pile.
     * Les éléments sont listés dans l'ordre de leur apparition, séparés par des espaces.
     *
     * @return une chaîne de caractères représentant la pile.
     */
    @Override
    public String toString() {
        String result = "";
        Node<T> current = head;
        while (current != null) {
            result += current.data;
            if (current.next != null) {
                result += " ";
            }
            current = current.next;
        }
        return result;
    }

    /**
     * Convertit la pile en un tableau d'objets.
     * Les éléments sont ordonnés du sommet vers le bas de la pile.
     *
     * @return un tableau contenant les éléments de la pile.
     */
```

```java
    public Object[] toArray() {
        int size = 0;
        Node<T> current = head;
        while (current != null) {
            size++;
            current = current.next;
        }

        Object[] array = new Object[size];
        current = head;
        for (int i = 0; i < size; i++) {
            array[i] = current.data;
            current = current.next;
        }
        return array;
    }

    /**
     * Crée un itérateur pour parcourir les éléments de la pile.
     *
     * @return un itérateur positionné au sommet de la pile.
     */
    public It<T> iterator() {
        return new It<>(head);
    }
}
```