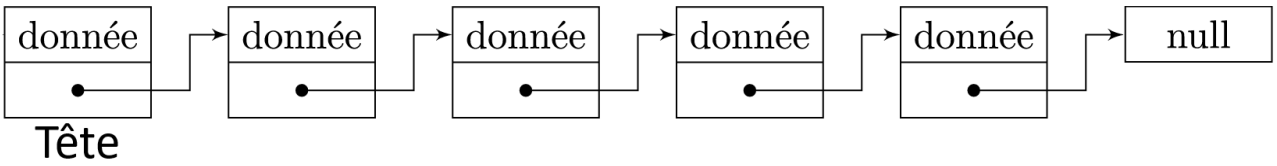
Liste chaînée

Une **liste chaînée** est une structure de données composée de plusieurs **maillons**. Chaque maillon contient deux informations :

1. une **valeur** (la donnée stockée) ;
2. une **référence** vers le maillon suivant de la liste.



La référence vers la suite pointe elle-même vers une liste chaînée, ce qui donne une définition récursive.

Vocabulaire :

- Maillon : un élément de la liste.
- Tête : le premier maillon de la liste.
- Queue : le reste de la liste (tous les maillons sauf le premier).
- Liste vide : une liste qui ne contient aucun maillon (référence `null`).

Remarques :

- La donnée peut être de n'importe quel type. Dans la suite du cours, on prendra des exemples avec des nombres.
- Contrairement aux tableaux, les listes chaînées ne sont pas stockées de façon contiguë en mémoire : chaque maillon peut être placé à un endroit différent.

Structure d'un maillon :

```java
public class Maillon {
    private int donnee;        // la donnée stockée
    private Maillon suivant;   // référence sur le prochain maillon
                               //(null si fin de liste)

    public Maillon(int donnee, Maillon suivant) {
        this.donnee = donnee;
        this.suivant = suivant;
    }

    public int getDonnee() { return donnee; }

```

```
13      public void setDonnee(int valeur) { this.donnee = valeur; }
14
15      public Maillon getSuivant() {  return suivant; }
16
17      public void setSuivant(Maillon suivant) {
18          this.suivant = suivant;
19      }
20  }
```

Structure d'une liste chaînée :

Une liste chaînée contient une référence vers le maillon de tête. À la construction, la liste est vide : la tête vaut `null`.

```
1   public class ListeChainee {
2       private Maillon tete; // null si liste vide
3
4       public ListeChainee() {
5           this.tete = null;
6       }
7
8       public boolean estVide() {
9           return tete == null;
10      }
11
12      // Réference de la tête
13      public Maillon getTete() {
14          return tete;
15      }
16  }
```

Ajout en tête :

Pour insérer une valeur v en tête de liste :

1. créer un nouveau maillon contenant v comme donnée et la tête actuelle comme suivant ;
2. mettre à jour la tête de la liste avec ce nouveau maillon.

Question : y a-t-il un problème si la liste est vide ? Réponse : non, car la tête vaut `null`, ce qui est accepté comme suivant dans le nouveau maillon.
Coût : $O(1)$ (opération constante).

```
1
2 public void ajouterEnTete(int v) {
3     Maillon nouveau = new Maillon(v, tete);
4     tete = nouveau;
5 }
```

Retrait de l'élément en tête de liste

Pour retirer l'élément en tête de liste, il suffit de modifier la référence de la tête pour la remplacer par le maillon suivant.
Question : que faire si la liste est vide ?
Coût : $O(1)$

```
1
2     // Retirer l'élément en tête
3     public void retirerTete() {
4         if (estVide()) {
5             throw new IllegalStateException("Liste vide");
6         }
7         tete = tete.getSuivant(); // devient null si un seul élément
8     }
```

Itération sur les éléments d'une liste

On peut parcourir les éléments d'une liste pour :

- les afficher (coût linéaire) ;
- rechercher la présence d'un élément (coût linéaire) ;
- rechercher la position d'un élément, c'est-à-dire obtenir une référence vers un maillon (coût linéaire).

Le parcours se fait en passant d'un maillon à son maillon suivant, ce qui peut aussi se définir de manière récursive.

```
1 public class IterateurListeChainee {
2     private final ListeChainee liste;
3     private Maillon ptr;
4
5     public IterateurListeChainee(ListeChainee liste) {
6         this.liste = liste;
7         this.ptr = liste.getTete(); // null si liste vide
8     }
```

```
9
10      // Maillon courant (peut être null si fin)
11      public Maillon courant() {
12          return ptr;
13      }
14
15      // Valeur du maillon courant
16      public int valeurCourante() {
17          if (fin()) throw new IllegalStateException("Fin de liste");
18          return ptr.getDonnee();
19      }
20
21      public boolean fin() {
22          return ptr == null;
23      }
24
25      public boolean debut() {
26          return ptr == liste.getTete();
27      }
28
29      public void avancer() {
30          if (fin()) throw new IllegalStateException("Fin de liste");
31          ptr = ptr.getSuivant(); // devient null si on dépasse
32      }
33  }
```

Remarque : reculer n'est pas possible dans une liste simplement chaînée. Pour se faire, il faut utiliser une liste doublement chaînée ou conserver une référence vers le maillon précédent.

Ajouter un élément après dans la liste d'un `IterateurListeChainee`

1. créer le nouveau maillon ;
2. mettre son *suivant* sur le *suivant* de l'itérateur ;
3. relier l'itérateur au nouveau maillon.

Coût : $O(1)$

```
1   public class IterateurListeChainee {
2
3       public void ajouterApres(int v) {
4           if (fin()) throw new IllegalStateException("Fin de liste");
5           Maillon nouveau = new Maillon(v, ptr.getSuivant());
6           ptr.setSuivant(nouveau);
7       }
```

Recherche d'un élément dans la liste d'un `IterateurListeChainee`

L'itérateur se déplace jusqu'à la première occurrence de la valeur recherchée, à partir de sa position actuelle. Si la valeur n'est pas présente, l'itérateur pointe sur `null`.

```
public class IterateurListeChainee {

    public void recherche(int v) {
        boolean trouve = false;
        while (!fin() && !trouve) {
            if (ptr.getDonnee() == v) {
                trouve = true;
            } else {
                avancer();
            }
        }
        // Si non trouvé, ptr vaut null après la boucle
    }
}
```

Méthode de recherche dans `ListeChainee` avec un itérateur

```
public class ListeChainee {

    public IterateurListeChainee recherche(int v) {
        IterateurListeChainee it = new IterateurListeChainee(this);
        it.recherche(v);
        return it.courant(); // pointe sur le maillon trouvé ou null
    }
}
```

Fonction d'affichage d'une liste chaînée

On peut écrire une fonction d'affichage qui utilise un itérateur pour parcourir la liste :

```
    public void affichage() {
        IterateurListeChainee it = new IterateurListeChainee(this);
        while (!it.fin()) {
            System.out.println(it.valeurCourante());
            it.avancer();
        }
    }
```