Collections : ensembles

Pratique de la programmation orientée-objet

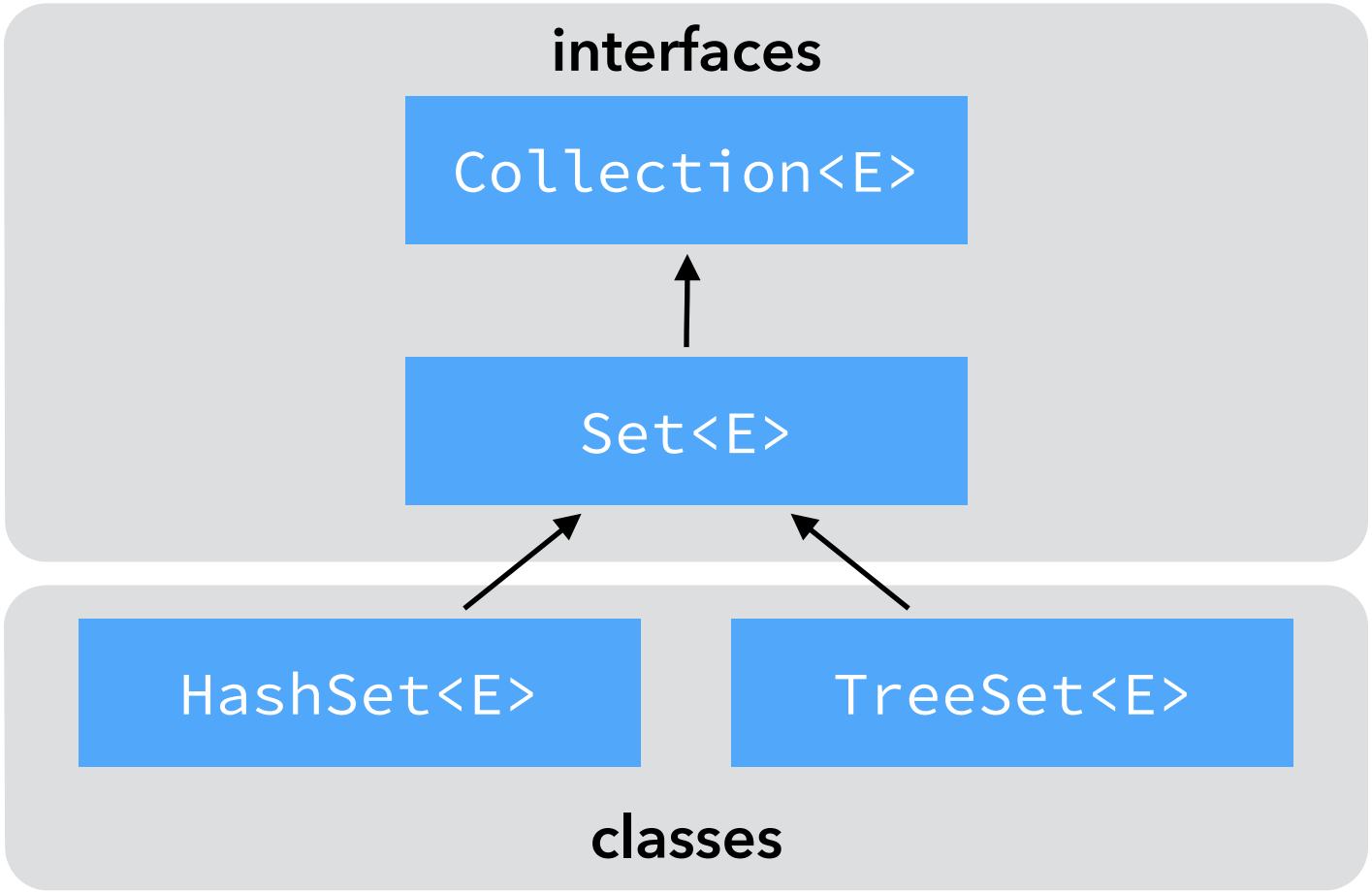
Michel Schinz – 2015-03-09

Collection n°3 : l'ensemble

Ensembles

Un **ensemble** (*set*) est une collection non ordonnée d'objets dans laquelle un objet ne peut apparaître qu'une fois au plus.
Cette notion d'ensemble correspond à la notion mathématique.

Ensembles de l'API Java



(D'autres mises en œuvre existent, mais sont rarement utilisées, donc ignorées ici.)

L'interface Set

Le concept d'ensemble est représenté dans l'API Java par l'interface `Set` du paquetage `java.util`. Tout comme `Collection` – dont elle hérite – cette interface est générique et son paramètre de type représente le type des éléments de l'ensemble :

```
public interface Set<E>
    extends Collection<E> {
  // … méthodes
}
```

Cette interface n'ajoute aucune méthode à celles héritées de `Collection`, son seul but étant d'offrir un type distinct pour les ensembles.

Ensembles mathématiques

A chaque opération importante sur les ensembles mathématiques correspond une méthode de l'interface `Set`, avec une différence importante : contrairement aux opérations mathématiques, les méthodes modifient l'ensemble auquel on les applique.

Opération	Méthode
union (∪)	`addAll`
test d'appartenance (∈?)	`contains`
test d'inclusion (⊆?)	`containsAll`
différence (\)	`removeAll`
intersection (∩)	`retainAll`

Ensembles immuables

Tout comme pour les listes et les tables associatives, la classe `Collections` offre des méthodes permettant de créer des ensembles immuables :

- `<E> Set<E> emptySet()` : retourne un ensemble vide immuable.
- `<E> Set<E> singleton(E e)` : retourne un ensemble contenant uniquement l'élément donné.

Vues non modifiables

Tout comme pour les listes et les tables associatives, la classe `Collections` offre une méthode permettant d'obtenir une vue non modifiable sur un ensemble :

```
<E> Set<E> unmodifiableSet(Set<E> s)
```

Comme toujours, prenez garde au fait qu'il s'agit d'une vue et que toute éventuelle modification à l'ensemble sous-jacent sera répercutée sur la vue ! Celle-ci est donc non modifiable mais pas forcément immuable.

Règle des ensembles imm.

Pour obtenir un ensemble immuable à partir d'un ensemble quelconque, obtenez une vue non modifiable d'une copie de cet ensemble.

La copie peut se faire au moyen du constructeur de copie de l'une des mises en œuvre. Par exemple, en utilisant `HashSet`, on obtient :

```
Set<…> immutableSet =
  Collections.unmodifiableSet(
    new HashSet<>(set));
```

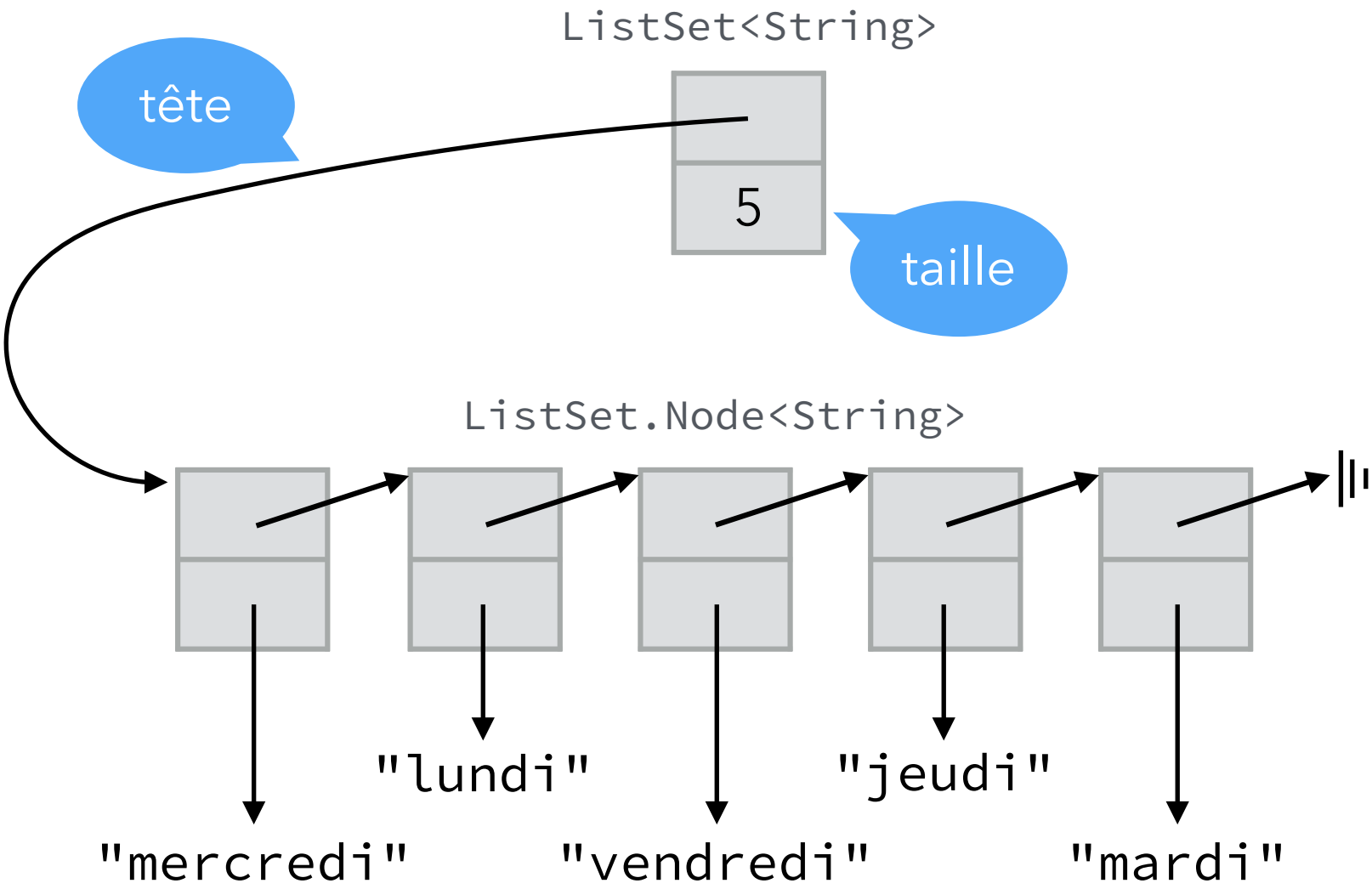
Mise en œuvre n° 1 : listes-ensembles

Listes-ensembles

Une simple liste peut être utilisée pour représenter un ensemble, pour peu que l'on s'assure de l'absence de doublons.
Cette représentation est simple à mettre en œuvre et à comprendre, mais très peu efficace, toutes les opérations importantes sur les ensembles étant en $O(n)$.
L'API Java ne fournit pas une telle mise en œuvre des ensembles. On peut néanmoins supposer, pour des raisons pédagogiques, qu'une telle mise en œuvre est fournie sous la forme d'une classe nommée `ListSet`.

ListSet

(Utilisant une liste simplement chaînée)



ListSet

Les ensembles représentés comme des listes sont, on l'a vu, peu efficaces car les principales opérations – ajout ou suppression d'un élément, test d'appartenance, etc. – ont une complexité de $O(n)$.

Toutefois, il n'est possible de faire mieux que si des opérations autres que l'égalité sont disponibles sur les éléments, à savoir :

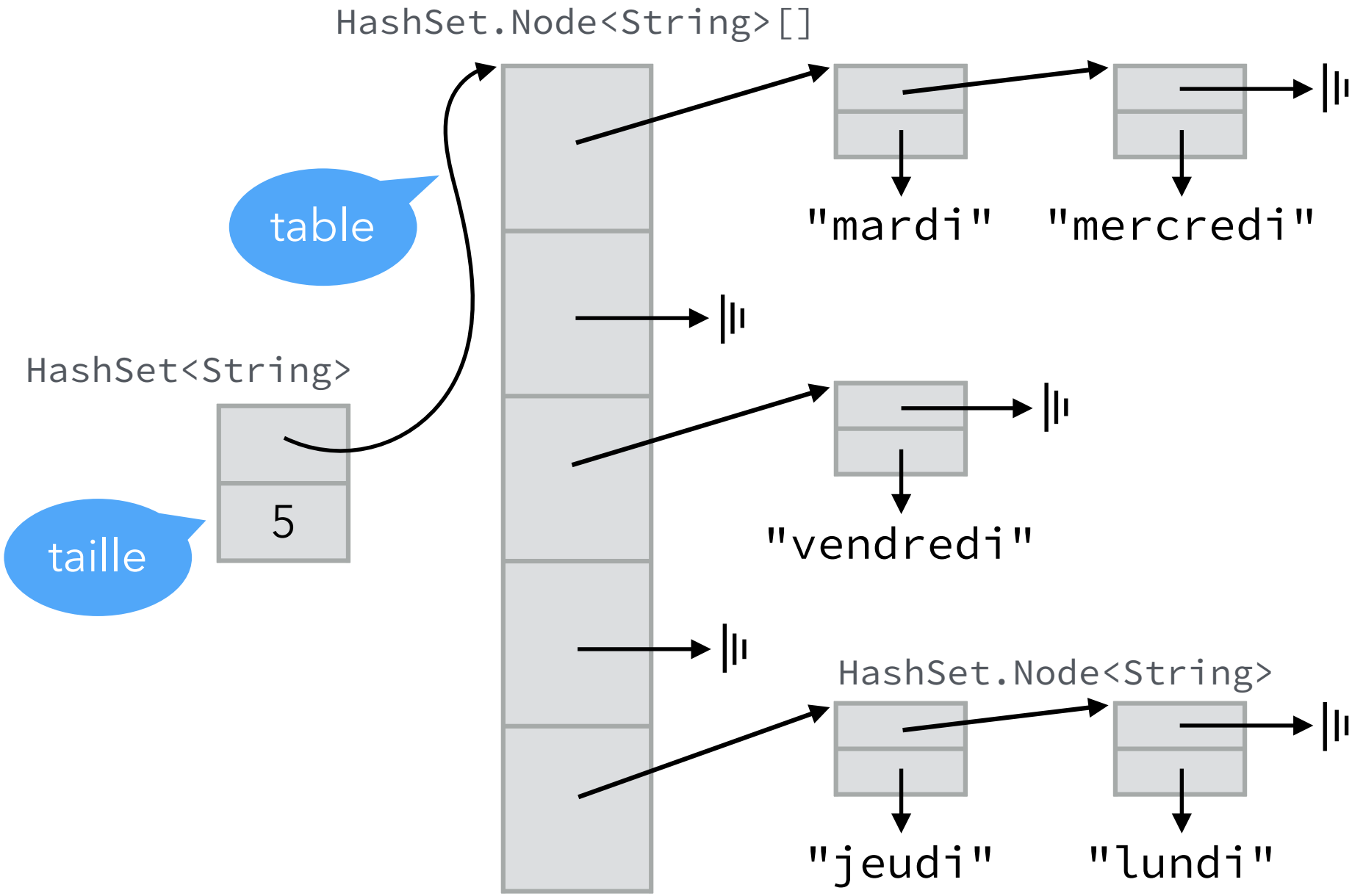
- un ensemble dont les éléments peuvent être hachés peut être représenté par une table de hachage,
- un ensemble dont les éléments peuvent être (totalement) ordonnés peut être représenté par un arbre binaire de recherche.

Mise en œuvre n°2 : tables de hachage

Ensembles par hachage

Si les éléments d'un ensemble peuvent être hachés, une table de hachage constitue une excellente représentation pour un ensemble.
Dans une telle table, la valeur de hachage d'un élément – ramenée à l'intervalle compris entre 0 et la taille de la table – est utilisée pour déterminer laquelle des listes de la table le contient.
Cette représentation est utilisée par la classe `HashSet` de l'API Java.

HashSet


HashSet.Node<String>[]
table
HashSet<String>
5
taille
"mardi"
"mercredi"
"vendredi"
HashSet.Node<String>
"jeudi"
"lundi"

HashSet

La mise en œuvre des ensembles au moyen de tables de hachage est très efficace si :

1. la fonction de hachage est en O(1), et
2. les listes stockées dans la table ont une longueur proche de 1.

Sous ces conditions, les principales opérations sur l'ensemble (ajout, test d'appartenance) sont en O(1) !

Cette caractéristique fait des tables de hachage la mise en œuvre la plus efficace qui soit des ensembles.

Mise en œuvre n°3 : arbres de recherche

TreeSet

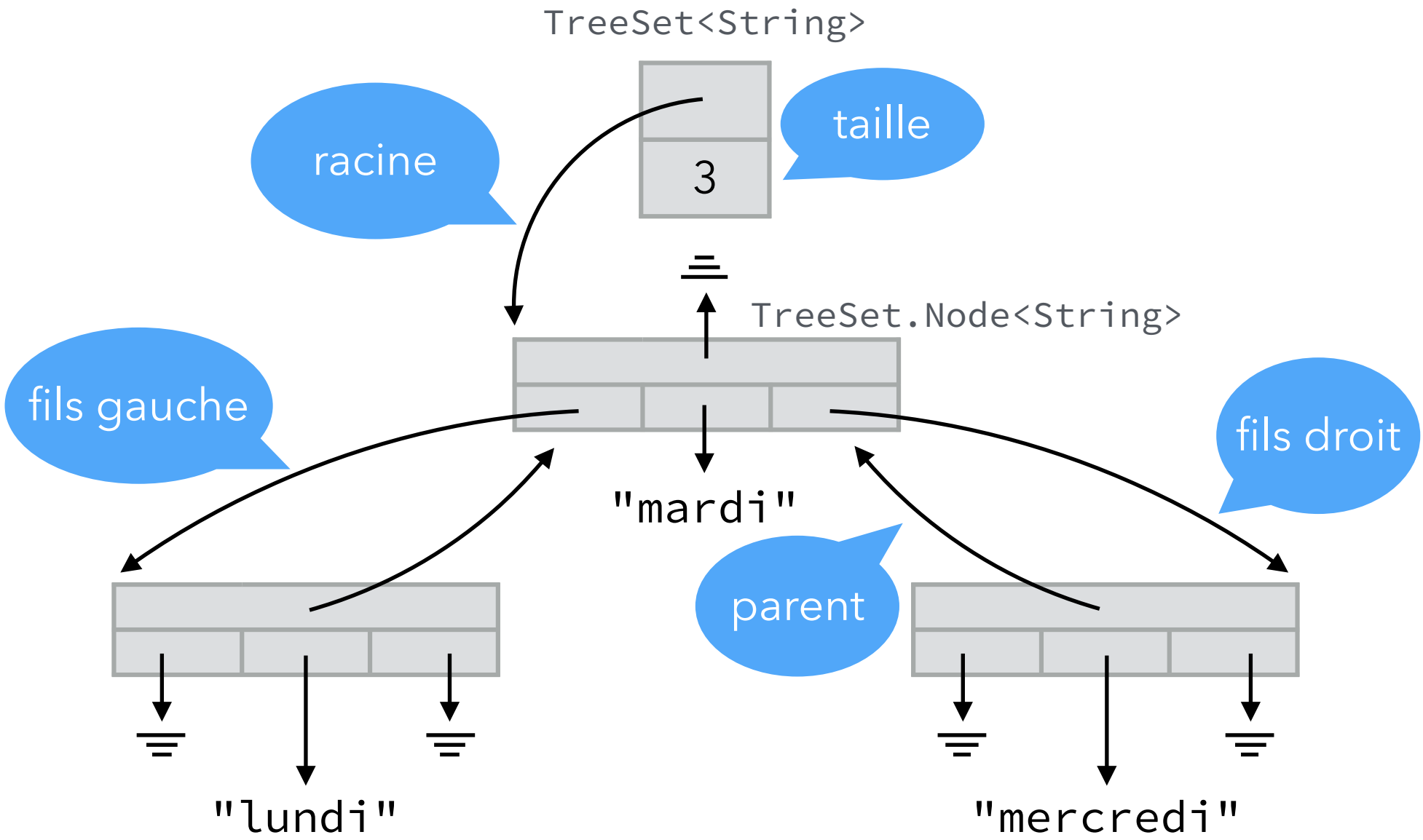
Un arbre de recherche peut être utilisé pour stocker les éléments d'un ensemble.

Les nœuds d'un tel arbre référencent les éléments de l'ensemble, et ils sont organisés de manière à ce que, pour chaque nœud :

- les éléments de tous les nœuds du sous-arbre gauche soient strictement plus petits que celui du nœud, et
- les éléments de tous les nœuds du sous-arbre droit soient strictement plus grands que celui du nœud.

La classe `TreeSet` de l'API Java utilise cette technique pour mettre en œuvre les ensembles.

TreeSet


TreeSet<String>
racine
taille
3
TreeSet.Node<String>
fils gauche
fils droit
"mardi"
parent
"lundi"
"mercredi"

TreeSet

La mise en œuvre des ensembles au moyen d'un arbre binaire de recherche est bien plus efficace que celle basée sur les listes, puisque les principales opérations – ajout, test d'appartenance – ont une complexité de $O(\log n)$ plutôt que $O(n)$.
Toutefois, étant donné que ces mêmes opérations ont une complexité de $O(1)$ avec les tables de hachage, ces dernières restent généralement préférables.
La classe `TreeSet` est surtout intéressante dans le cas où il est important de pouvoir parcourir les éléments dans l'ordre.

Règle HashSet/TreeSet

Utilisez `HashSet` comme mise en œuvre des ensembles en Java, sauf lorsqu'il est utile de parcourir les éléments en ordre croissant, auquel cas vous pourrez préférer `TreeSet`.

`HashSet` a d'une part l'avantage d'être généralement plus rapide que `TreeSet`, et d'autre part celui de ne pas demander à ce que les éléments de l'ensemble puissent être ordonnées.

Ensembles/tables assoc.

On l'a vu, il y a une grande similarité entre les ensembles et les tables associatives :

- un ensemble peut être vu comme une table associative dont seules les clefs sont prises en compte, les valeurs étant ignorées,
- une table associative peut être vu comme un ensemble de paires clef/valeur – pour peu que la valeur soit ignorée dans les tests d'égalité, les comparaisons et le hachage.

Pour cette raison, dans la bibliothèque Java, `HashSet` est mis en œuvre au moyen de `HashMap`, et `TreeSet` au moyen de `TreeMap`.