Fiche — structures de données Java (opérations de base)

Notation : n = nombre d'éléments. $\Theta(1)$ amorti = constant en moyenne sur une suite d'opérations.

Structure	Accès indexé	Ajout fin	Ajout tête	Insertion milieu	Suppr. fin	Suppr. tête	Rech. `contains`	Remarques
`T[]`	$\Theta(1)$	—	—	$\Theta(n)$	—	—	$\Theta(n)$	Taille fixe, décalages coûteux
`ArrayList`	$\Theta(1)$	$\Theta(1)$*	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	Séquence dynamique
`LinkedList`	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$**	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	**itérateur déjà positionné
`ArrayDeque`	—	$\Theta(1)$*	$\Theta(1)$*	—	$\Theta(1)$*	$\Theta(1)$*	$\Theta(n)$	Excellent pour pile/file

`*` : $\Theta(1)$ amorti, c'est à dire constant en moyenne sur une suite d'opérations.

Ensembles (Set)

Structure	`add`	`remove`	`contains`	Itération	Ordre
`HashSet`	$\Theta(1)$**	$\Theta(1)$*	$\Theta(1)$*	$\Theta(n)$	Aucun
`LinkedHashSet`	$\Theta(1)$**	$\Theta(1)$*	$\Theta(1)$*	$\Theta(n)$	Insertion
`TreeSet`	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(n)$	Trié

`*` : $\Theta(1)$ en moyenne (pire cas $\Theta(n)$ si la fonction de hachage ne disperse pas les clés)

`**` : $\Theta(1)$ en moyenne, pire cas $\Theta(n)$, et amorti à cause du re-hash.

Dictionnaires (Map)

Structure	`get`	`put`	`remove`	`containsKey`	Itération	Ordre
`HashMap`	$\Theta(1)$*	$\Theta(1)$**	$\Theta(1)$*	$\Theta(1)$*	$\Theta(n)$	Aucun
`LinkedHashMap`	$\Theta(1)$*	$\Theta(1)$**	$\Theta(1)$*	$\Theta(1)$*	$\Theta(n)$	Insertion
`TreeMap`	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(n)$	Trié

`*` : $\Theta(1)$ en moyenne (pire cas $\Theta(n)$ si la fonction de hachage ne disperse pas les clés)

`**` : $\Theta(1)$ en moyenne, pire cas $\Theta(n)$, et amorti à cause du re-hash.