

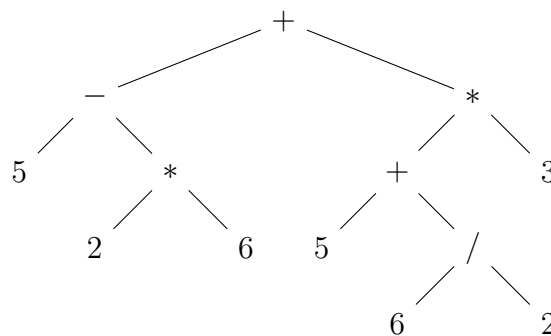
TP n° 5


Itérateurs et parcours d'arbre


L'objectif de cette séance de travaux pratiques est de comprendre le principe du parcours des éléments d'une structure arborescente et de mettre en place un itérateur en Java.


Exercice 1 – Parcours d'arbre binaire

En réutilisant le TAD Arbre Binaire du TP précédent, écrire l'algorithme des fonctions suivantes et tester à l'aide de l'arbre ci-dessous :



 Écrire la fonction `parcoursProfondeurInfixe` qui écrit dans la console les valeurs des noeuds dans l'ordre de leur parcours en profondeur infixe.


 Écrire la version itérative de la fonction `parcoursProfondeurPréfixé` qui retourne une `ArrayList<Object>` contenant les valeurs des noeuds dans l'ordre de leur parcours en profondeur préfixé. On utilisera le TAD Pile du TP précédent (voir annexe)

 Écrire la version itérative de la fonction `parcoursLargeur` qui retourne une `ArrayList<Object>` contenant les valeurs des noeuds dans l'ordre de leur parcours en largeur. On utilisera le TAD File du TP précédent (voir annexe).

Exercice 2 – Itérateur Java


Un itérateur est un objet capable d'itérer sur les valeurs d'une collection, c'est à dire qu'il peut prendre successivement toutes les valeurs d'une collection. En java, une structure de données peut être parcourue à l'aide d'un itérateur si elle implémente l'interface `Iterable`. Le cas échéant, il


est possible d'obtenir l'itérateur de la collection grâce à la méthode `iterator()` qui retourne un objet de type `Iterator<E>`.

 Créer une liste d'entiers remplie avec 10 entiers tirés aléatoirement dans l'intervalle $[0,10]$. Itérer sur les éléments de la liste en supprimant les éléments pairs au fur et à mesure du parcours.

Exercice 3 – Itérateur d'arbre binaire

Nous souhaitons désormais implémenter notre propre itérateur capable de parcourir tous les éléments d'un arbre binaire.

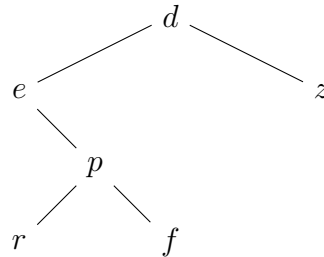
 Créer une nouvelle classe `ArbreBinaireItérateur` qui dérive de la classe `ArbreBinaire` et qui implémente l'interface `Iterable`.

 Créer une nouvelle classe `ABIterator` qui dérive de la classe `Iterator<Object>`. Implémenter le constructeur ainsi que les méthodes `next()` et `hasNext()` permettant de rendre l'itérateur opérationnel (Ne pas implémenter la méthode `remove()`). L'ordre du parcours des éléments (largeur ou profondeur d'abord) n'a pas d'importance.

 Tester l'itérateur en affichant successivement les nœuds de l'arbre de l'exercice 1 dans la console.

Annexe

Pour parcourir un arbre binaire de manière itérative, on a recours à une structure de données pour conserver, durant le traitement d'un élément, les éléments restant à traiter. Les figures suivantes illustrent les étapes nécessaires pour parcourir l'arbre binaire ci-dessous en profondeur d'abord (on utilise une pile) et en largeur d'abord (on utilise une file).



Parcours profondeur préfixé d'abord avec une pile

t_0	t_1	t_2	t_3	t_4	t_5	t_6
<div style="border: 1px solid black; padding: 5px; text-align: center;">d</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">e z</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">p z</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">r f z</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">f z</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">z</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;"></div>
empiler(d)	dépiler(d)	dépiler(e)	dépiler(p)	dépiler(r)	dépiler(f)	dépiler(z)
	traiter(d)	traiter(e)	traiter(p)	traiter(r)	traiter(f)	traiter(z)
	empiler(z)	empiler(p)	empiler(f)			
	empiler(e)		empiler(r)			

Parcours largeur d'abord avec une file

t_0	t_1	t_2	t_3	t_4	t_5	t_6
<div style="border: 1px solid black; padding: 5px; text-align: center;">d</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">z e</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">p z</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">p</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">f r</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;">f</div>	<div style="border: 1px solid black; padding: 5px; text-align: center;"></div>
enfiler(d)	défiler(d)	défiler(e)	défiler(z)	défiler(p)	défiler(r)	défiler(f)
	traiter(d)	traiter(e)	traiter(z)	traiter(p)	traiter(r)	traiter(f)
	enfiler(e)	enfiler(p)		enfiler(r)		
	enfiler(z)			enfiler(f)		