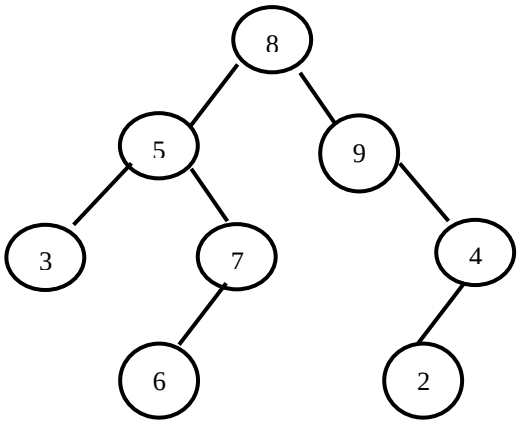
Les parcours d'arbres

Exercices obligatoires

A Exercices préliminaires

Les solutions de ces exercices se trouvent dans le document A1_A2_A3Sol sur moodle.

A1 Pour chacun des parcours, donnez l'ordre dans lequel vont apparaître les entiers de l'arbre suivant :



a) Parcours pré-ordre :

b) Parcours in-ordre

c) Parcours post-ordre :

d) Parcours par niveau :

A2 Dans la classe *ExpressionArithmetique* que vous avez complétée la semaine dernière, les méthodes ont été écrites de façon récursive.
Si vous deviez les réécrire de façon itérative, quel type de parcours choisiriez-vous ?

a) La méthode `resultat()`

b) La méthode `notationInfixe()`

A3 Dans un ABR, quel parcours faut-il choisir si on veut explorer les éléments par ordre croissant ?

B Implémentation des itérateurs

La classe *ArbreDEntiers* a été complétée par 4 itérateurs.
On vous demande d'implémenter ces 4 itérateurs.
La classe *TestIterateurs* permet de les tester avec l'arbre de l'exercice A.

B1 Commencez par implémenter l'itérateur en pré-ordre.
Pour celui-ci, il vous reste à compléter les méthodes `remplirFile()`,`hasNext()` et `next()` de la classe interne *PreIterator*.
La classe *PreIterator* possède un attribut : une file d'entiers (*ArrayDeque<Integer>).*
Le constructeur de la classe va s'occuper de remplir cette file avec tous les entiers contenus dans l'arbre.
La méthode `hasNext()` vérifie si la file est non vide.
La méthode `next()` « défile ».
Le but de cet itérateur est de parcourir l'arbre en pré-ordre !
Il faut donc « enfiler » les objets dans la file de façon à respecter ce parcours. C'est la méthode `remplirFile()` qui se charge de remplir la file. Il s'agit d'une méthode récursive !

B2 Ajoutez les itérateurs en in-ordre et en post-ordre.
Ces 2 itérateurs s'implémentent de façon similaire à l'itérateur en pré-ordre.
Ils utilisent aussi une file.
Seule la méthode `remplirFile()` change !

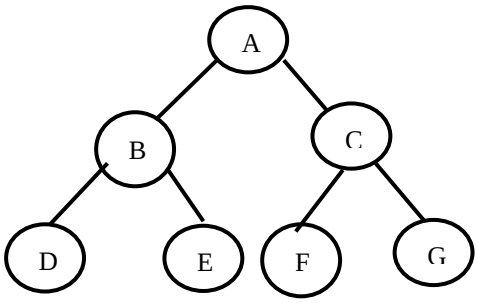
Remarque :
L'itérateur in-ordre a été sélectionné comme itérateur par défaut :

```
public Iterator<Integer> iterator (){
}
```

C'est celui-ci qui sera utilisé dans un *foreach* !

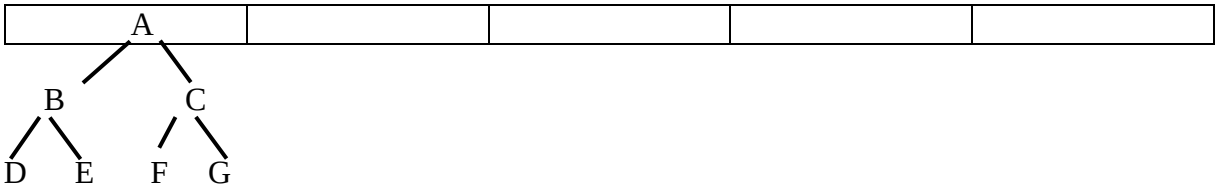
B3 Ajoutez l'itérateur par niveau.
Attention ! Cet itérateur s'implémente aussi via une file mais de façon différente !
La file est une file de nœuds.
Le constructeur, si l'arbre est non vide, n'y place que la racine.
La méthode `hasNext()` vérifie si la file est non vide.
La méthode `next()` « défile » un nœud et «enfile » les racines de ses fils **non vides**.

Exemple :

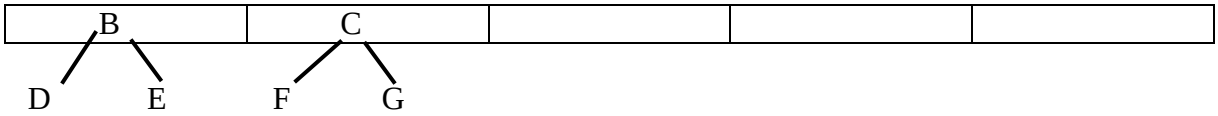


File :

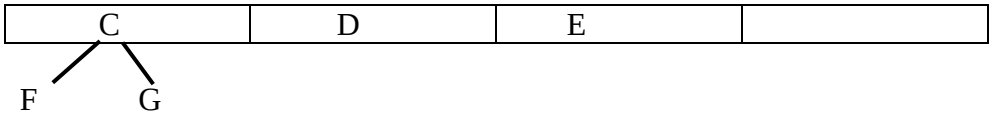
Au départ :



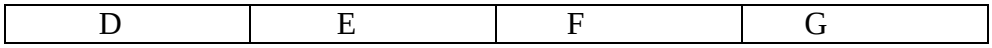
Après un next() :



Après un next() :



Après un next() :

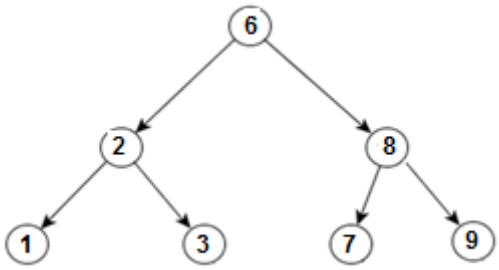


Etc…

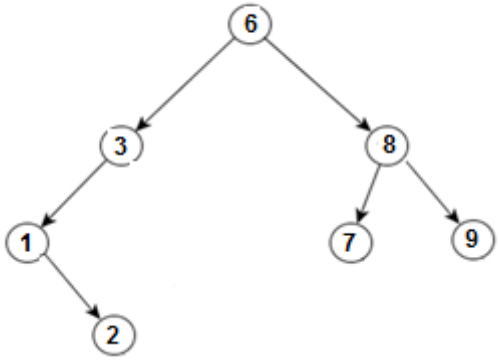
💣Exercice super défi💣

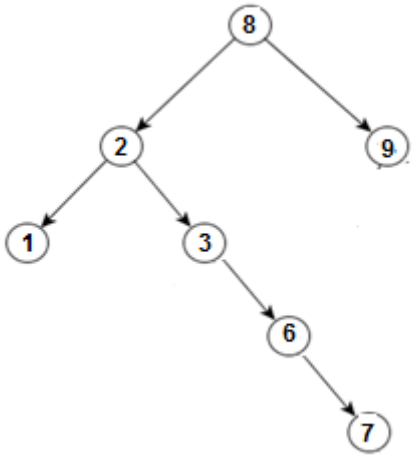
Dans la classe *ABRDEntiers*, on vous demande d'écrire une méthode qui construit un arbre très équilibré qui contient les mêmes valeurs que l'arbre courant.
(Cette méthode est surtout intéressante pour les ABR.)

Exemple d'un arbre équilibré :

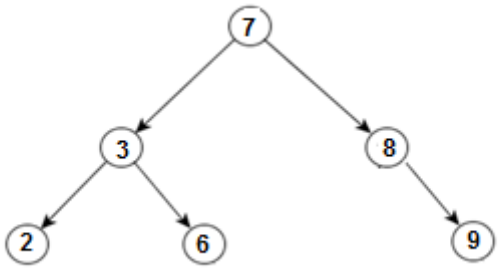


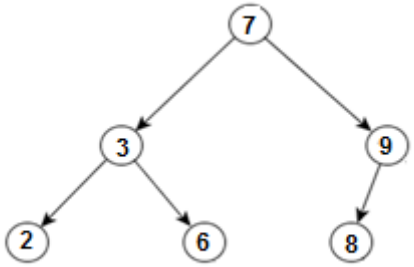
Tous les arbres différents de celui-ci qui contiennent les mêmes entiers sont non équilibrés :

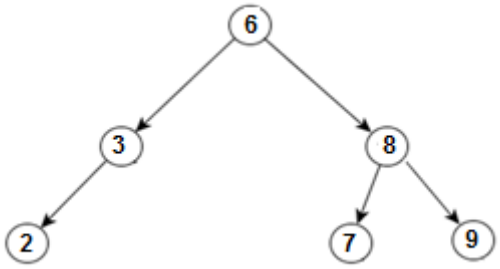


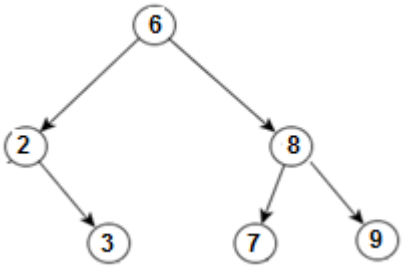


Etc…
Il n'existe qu'une solution d'arbre équilibré pour un arbre qui possède 1, 3, 7, 15, … entiers.
Cet arbre équilibré est dit « complétement rempli ».
Prenons le cas d'un arbre qui possède 6 entiers : 2, 3, 6, 7, 8 et 9 :
Les arbres suivants sont équilibrés :

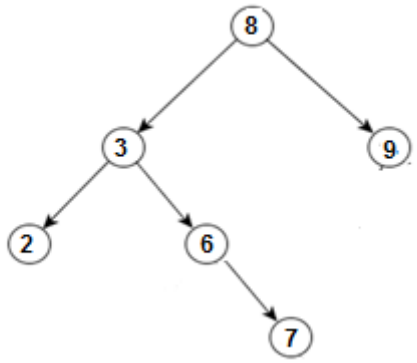


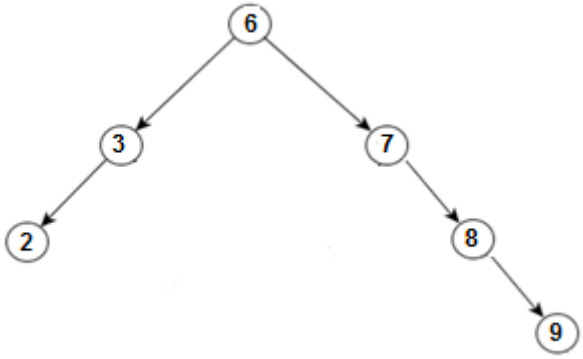






Tous les autres arbres ne le sont pas :





Etc.

Vous allez écrire la méthode récursive `construit(int n, Iterator it)`
Celle-ci sera appelée par la méthode `equilibre()` avec un itérateur en in-ordre.
La classe *TestEquilibre* permet de tester cette méthode.