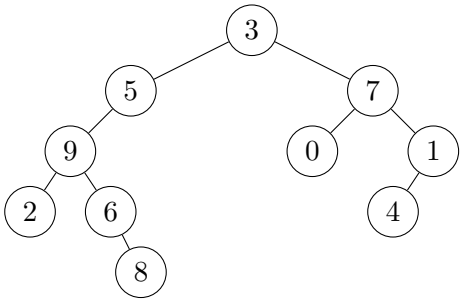
Pendant tout ce tp, nous allons prendre comme exemple l'arbre suivant.



Commençons par définir différentes façons de parcourir un arbre binaire :

1. **Parcours infixe** : on parcourt le sous-arbre de gauche dans l'ordre infixe, on visite le sommet puis on parcourt le sous-arbre de droite dans l'ordre infixe.
   Pour l'arbre précédent ce parcours est : 2 9 6 8 5 3 0 7 4 1.
2. **Parcours préfixe** : on visite le sommet, on parcourt le sous-arbre de gauche dans l'ordre préfixe puis on parcourt le sous-arbre de droite dans l'ordre préfixe.
   Pour l'arbre précédent ce parcours est : 3 5 9 2 6 8 7 0 1 4.
3. **Parcours postfixe** : on parcourt le sous-arbre de gauche dans l'ordre postfixe, on parcourt le sous-arbre de droite dans l'ordre postfixe puis on visite le sommet.
   Pour l'arbre précédent ce parcours est : 2 8 6 9 5 0 4 1 7 3.

On considère ainsi la classe `Noeud` suivante :

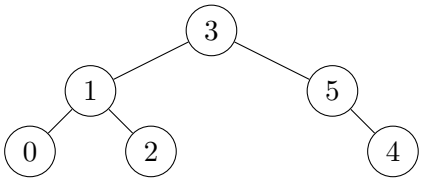
```
public class Noeud {
    private int etiquette;
    private Noeud gauche;
    private Noeud droit;

    public Noeud(int etiquette, Noeud g, Noeud d) {
        this.etiquette = etiquette;
        this.gauche = g;
        this.droit = d;
    }

    public Noeud(int etiquette) {
        this(etiquette,null,null);
    }
}
```

1. Définir une méthode `public void afficheInfixe()` dans la classe `Noeud` qui affiche l'arbre commençant à `this` dans l'ordre infixe.
2. Tester dans une classe `Main` la méthode `afficheInfixe()` de la classe `Noeud` sur l'exemple donné en introduction grâce au code suivant :

```
public class Main{

        public static void main(String[] args){

                Noeud a = new Noeud(6,null,new Noeud(8));
                Noeud b = new Noeud(9, new Noeud(2), a);
                Noeud c = new Noeud(5, b, null);
                Noeud d = new Noeud(1, new Noeud(4), null);
                Noeud e = new Noeud(7, new Noeud(0), d);
                Noeud f = new Noeud(3, c, e);

                f.afficheInfixe();
        }
}
```

3. Définir dans la classe `Noeud` des méthodes `public void affichePrefixe()` et `public void affichePostfixe()` qui affichent `this` respectivement dans l'ordre préfixe et suffixe.
4. Définir dans la classe `Noeud` une méthode `int nbDeNoeuds()` qui retourne le nombre de nœuds de `this`.
5. Définir dans la classe `Noeud` une méthode `int somme()` qui retourne la somme des étiquettes de `this`.
6. Définir dans la classe `Noeud` une méthode `int profondeur()` qui retourne la profondeur de `this`. La profondeur d'un arbre est la longueur de sa plus longue branche ; par exemple la profondeur de l'arbre-exemple est 4 : sa plus longue branche est $3-5-9-6-8$.
7. Définir dans la classe `Noeud` une méthode `boolean recherche(int e)` qui renvoie `true` si un nœud de l'arbre est étiqueté par `e`.
8. Définir un constructeur `Noeud(Noeud arbre)` qui crée une **copie** de l'arbre donné en argument.
9. (facultatif) Définir un nouveau constructeur `Noeud(int[] tab)` qui prend en entrée un tableau non vide de taille `n`.
   — Soit `r` la moitié de `n` (arrondi à l'inférieur : r = n/2).
   — La racine de `this` a pour étiquette tab[r].
   — Soit `tabG =` [`tab`[0], ..., `tab`[$r-1$]]. Le sous-arbre de gauche est `new Noeud(tabG)` et
   — soit `tabD =` [`tab`[$r+1$], ..., `tab`[$l-1$]] alors le sous-arbre de droite est `new Noeud(tabD)`.

   Par exemple `new Noeud`([0, 1, 2, 3, 4, 5]) est :



10. (facultatif) Tester le constructeur sur l'arbre précédent via le code :

```
int[] tab = {0,1,2,3,4,5};

Noeud g = new Noeud(tab);

g.afficheInfixe();
g.affichePrefixe();
```

Ce dernier doit afficher 0 1 2 3 4 5 et 3 1 0 2 5 4 .