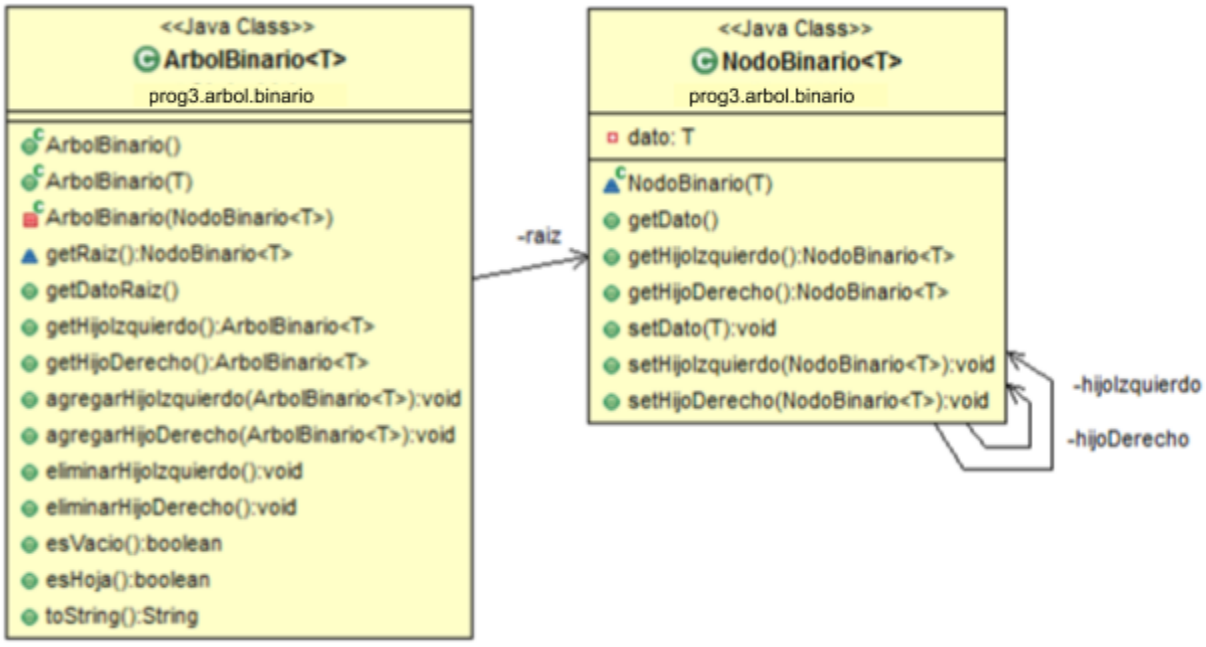
Programación III
TEMA 4: Árboles binarios
Práctica nº 4 - A

Puede continuar trabajando en su proyecto Programacion3. El archivo zip descargado desde la página de la cátedra no es un proyecto eclipse, por tanto:

1. descomprima el archivo zip
2. sobre la carpeta **src** de su proyecto Programacion3 haga click con el botón derecho del mouse y seleccione la opción *Import* > *FileSystem*.
3. Haga click en "Browse" y busque la carpeta descomprimida y seleccione la carpeta **src** (haga click para que aparezca el check seleccionado)
4. Haga click en el botón finalizar

1. Considere la siguiente especificación de la clase ArbolBinario (con la representación hijo izquierdo e hijo derecho).



- El constructor **ArbolBinario()** inicializa un árbol binario vacío, es decir, la raíz en null.
- El constructor **ArbolBinario(T dato)** inicializa un árbol que tiene como raíz un nodo binario. Este nodo tiene el dato enviado como parámetro y ambos hijos nulos.
- El constructor **ArbolBinario(NodoBinario<T> nodo)** inicializa un árbol donde el nodo pasado como parámetro es la raíz. (Notar que **NO** es un método público).
- El método **getRaiz():NodoBinario<T>**, retorna el nodo ubicado en la raíz del árbol. (Notar que **NO** es un método público).
- El método **getDatoRaiz():T**, retorna el dato almacenado en el NodoBinario raíz del árbol.
- Los métodos **getHijoIzquierdo():ArbolBinario<T>** y **getHijoDerecho():ArbolBinario<T>**, retornan los hijos izquierdo y derecho

respectivamente de la raíz del árbol. Tenga en cuenta que los hijos izquierdo y derecho del NodoBinario raíz del árbol son NodosBinarios y usted debe devolver ArbolesBinarios, por lo tanto debe usar el constructor privado **ArbolBinario (NodoBinario<T> nodo)** para generar el árbol binario correspondiente.

- El método **agregarHijoIzquierdo(ArbolBinario<T> unHijo)** y **agregarHijoDerecho(ArbolBinario<T> unHijo)** agrega un hijo como hijo izquierdo o derecho del árbol. Tenga presente que unHijo es un ArbolBinario y usted debe enganchar un NodoBinario como hijo. Para ello utilice el método privado getRaiz.
- El método **eliminarHijoIzquierdo()** y **eliminarHijoDerecho()**, eliminan el hijo correspondiente NodoBinario raíz del árbol receptor.

Analice la implementación en JAVA de las clases ArbolBinario y NodoBinario brindadas por la cátedra.

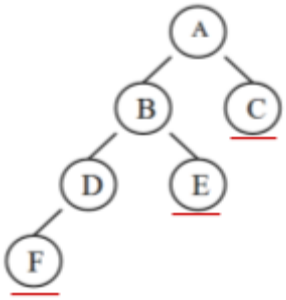
a. Realice el diagrama del siguiente árbol. En particular indique **cómo quedan representados los nodos que son HOJA. ¿Cómo se representa el hijo izquierdo y el hijo derecho de una HOJA?**

```
ArbolBinario<Integer> arbolBinarioB=new ArbolBinario<Integer>(1);
ArbolBinario<Integer> hijoIzquierdoB=new ArbolBinario<Integer>(2);
hijoIzquierdoB.agregarHijoIzquierdo(new ArbolBinario<Integer>(3));
hijoIzquierdoB.agregarHijoDerecho(new ArbolBinario<Integer>(4));
ArbolBinario<Integer> hijoDerechoB=new ArbolBinario<Integer>(6);
hijoDerechoB.agregarHijoIzquierdo(new ArbolBinario<Integer>(7));
hijoDerechoB.agregarHijoDerecho(new ArbolBinario<Integer>(8));
arbolBinarioB.agregarHijoIzquierdo(hijoIzquierdoB);
arbolBinarioB.agregarHijoDerecho(hijoDerechoB);
```

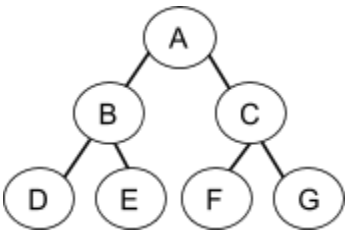
2. Considere la clase **Recorrido** brindada por la cátedra. Implemente los métodos correspondientes a los 3 tipos de recorrido en profundidad: **PreOrder**, **InOrder** y **PostOrder**.

3. Agregue a la clase Arbol Binario los siguientes métodos (Implemente, luego en el punto 4 podrá probarlos)
   a. **frontera():ListaGenerica<T>** Se define **frontera** de un árbol binario, a las hojas de un árbol binario recorridos de izquierda a derecha.
   NOTA: analice los 3 tipos de recorridos en profundidad de un ArbolBinario y elija el que corresponde.

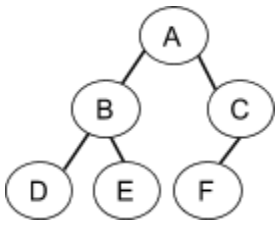
   Ejemplo: para el árbol binario del gráfico, el resultado será una lista conteniendo los valores: **F, E, C**



b. **lleno()**: **boolean.** Devuelve true si el árbol es lleno. Un árbol binario es lleno si tiene todas las hojas en el mismo nivel y además tiene todas las hojas posibles (es decir todos los nodos intermedios tienen dos hijos).

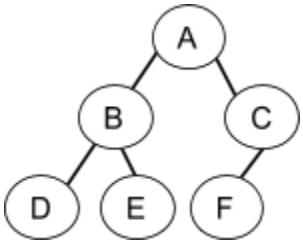


Éste árbol binario **ES** lleno

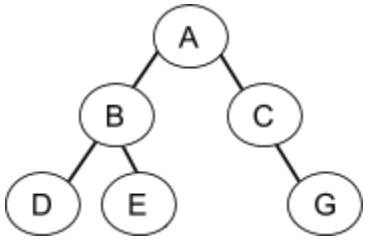


Éste árbol binario **NO ES** lleno

c. Indique cual sería la lógica de la solución (no implemente), para el siguiente método:
completo(): boolean. Devuelve true si el árbol es completo. Un árbol binario de altura h es completo si es lleno hasta el nivel (h-1) y el nivel h se completa de izquierda a derecha.



Éste árbol binario **ES** completo



Éste árbol binario **NO ES** completo

4. JUnit (prueba de la implementación de los ejercicios anteriores)

a. Descargue del sitio https://github.com/junit-team/junit/releases el achivo .jar (librería recomendada version 4.7) correspondiente a JUnit ó descarguelo de la página de la cátedra.
b. Incluya dicha librería en su proyecto (cree una carpeta lib de modo que la librería quede dentro de su proyecto)
c. Ejecute la clase ArbolBinarioTest y verifique que los Test se ejecutan exitosamente.