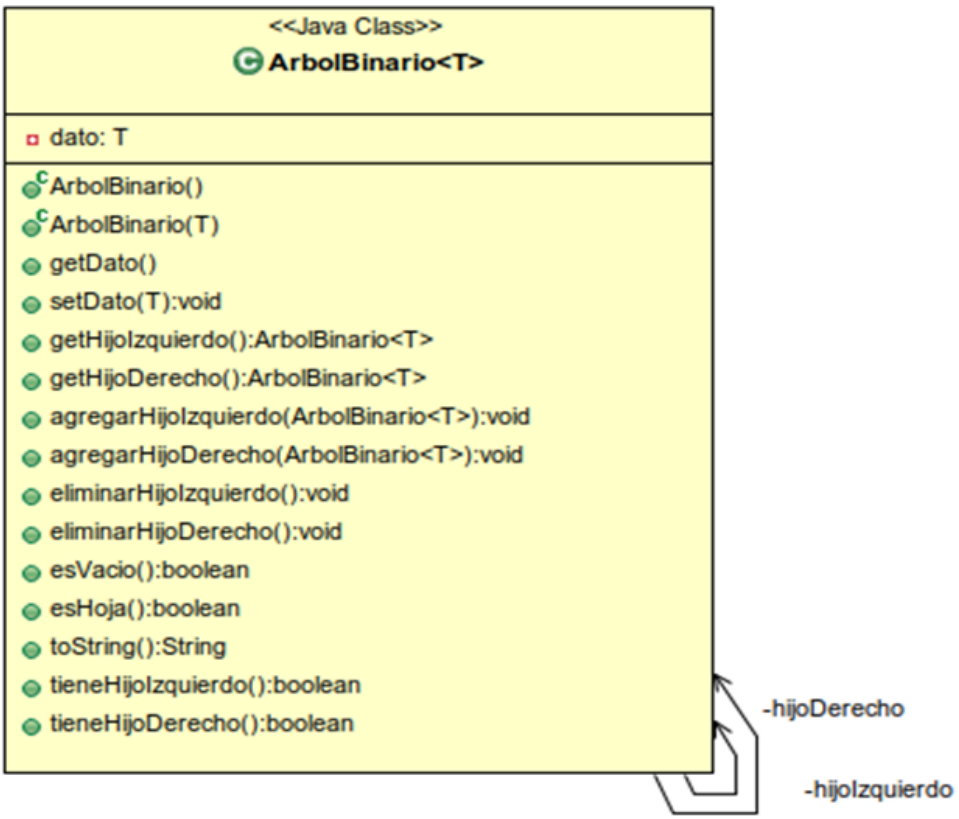
Práctica 3
Árboles Binarios y Árboles Binarios de Búsqueda
AB, ABB, recorridos y ejercicios de aplicación

Importante: Descargue el material en archivo .zip disponible del Campus. Se recomienda continuar trabajando dentro del proyecto AyED y generar paquetes y subpaquetes para esta práctica. El archivo .zip contiene las clases que deberá importar al proyecto dentro del IDE con el cual trabaja habitualmente.

1. Considere la siguiente especificación de la clase **ArbolBinario**



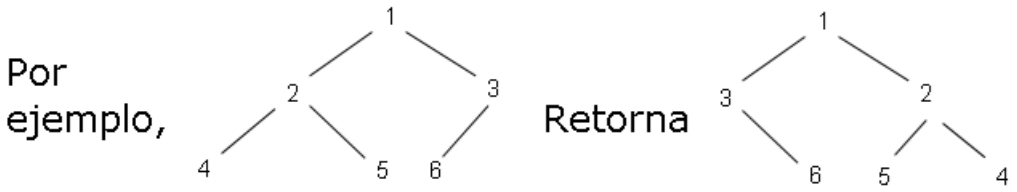
- El constructor **ArbolBinario(T dato)** inicializa un árbol con el dato pasado como parámetro y ambos hijos nulos.
- Los métodos **getHijoIzquierdo():ArbolBinario<T>** y **getHijoDerecho():ArbolBinario<T>**, retornan los hijos izquierdo y derecho respectivamente del árbol.
- El método **agregarHijoIzquierdo(ArbolBinario<T> unHijo)** y **agregarHijoDerecho(ArbolBinario<T> unHijo)** agrega un hijo como hijo izquierdo o derecho del árbol.
- El método **eliminarHijoIzquierdo()** y **eliminarHijoDerecho()**, eliminan el hijo correspondiente.
- El método **esVacio()** indica si el árbol está vacío y el método **esHoja()** no tiene hijos.
- El método **tieneHijoIzquierdo()** y **tieneHijoDerecho()** devuelven un booleano indicando si tiene dicho hijo el árbol receptor del mensaje.

Analice la implementación en JAVA de la clase **ArbolBinario** brindada por la cátedra.

2. Agregue a la clase **ArbolBinario** los siguientes métodos:

a) **public int contarHojas()** Devuelve la cantidad de árbol/subárbol hojas del árbol receptor.

b) **public ArbolBinario<T>espejo()** Devuelve el árbol binario espejo del árbol receptor. Por ejemplo:



c) **public void entreNiveles(int n, int m)** Imprime por consola el recorrido por niveles de los elementos del árbol receptor entre los niveles n y m (ambos inclusive). (0≤n<m≤altura del árbol)

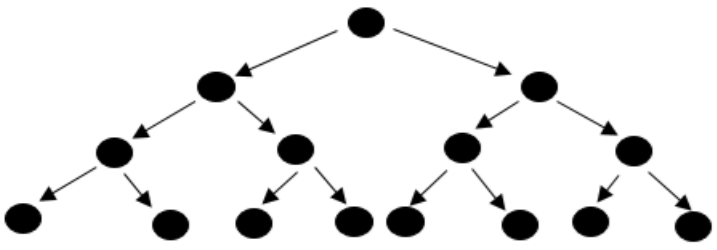
3. Defina una clase Java denominada **ContadorArbol** cuya función principal es proveer métodos de validación sobre árboles binarios de enteros. Para ello la clase tiene como variable de instancia un **ArbolBinario<Integer>**. Implemente en dicha clase un método denominado **numerosPares()** que devuelve en una estructura adecuada (sin ningún criterio de orden) todos los elementos pares del árbol (divisibles por 2).

Defina la clase dentro del paquete **tp03.ejercicio3**

a) Implemente el método realizando un recorrido InOrden.

b) Implemente el método realizando un recorrido PostOrden.

4. Una red binaria es una red que posee una topología de árbol binario lleno. Por ejemplo:



Los nodos que conforman una red binaria llena tiene la particularidad de que todos ellos conocen cuál es su retardo de reenvío. El retardo de reenvío se define como el período comprendido entre que un nodo recibe un mensaje y lo reenvía a sus dos hijos.

Su tarea es calcular el mayor retardo posible, en el camino que realiza un mensaje desde la raíz hasta llegar a las hojas en una red binaria llena.

Nota: asuma que cada nodo tiene el dato de retardo de reenvío expresado en cantidad de segundos.

a) Indique qué estrategia (recorrido en profundidad o por niveles) utilizará para resolver el problema.

b) Cree una clase Java llamada **RedBinariaLlena,** dentro del paquete **tp03.ejercicio4,** donde deberá implementar lo solicitado en el método

public int retardoReenvio()

5. Implemente una clase Java llamada **ProfundidadDeArbolBinario** que tiene como variable de instancia un árbol binario de números enteros y un método de instancia **public int sumaElementosProfundidad(int p)** el cuál devuelve la suma de todos los nodos del árbol que se encuentren a la profundidad pasada como argumento.

Defina la clase dentro del paquete **tp03.ejercicio5**

6. Considere la siguiente especificación de la clase **ArbolBinarioDeBusqueda** (con la representación hijo izquierdo e hijo derecho):

Aclaración:

En la imagen no aparece la definición de la variable de tipo en la clase ArbolBinarioDeBusqueda. La misma es **<T extends Comparable<T>>**, lo cual indica que la clase representada por la variable T implemente la interface Comparable.
Las clases que implementan la interface ***java.lang.Comparable<T>*** permiten que sus instancias se puedan comparar entre sí. Para lograr esto, deben implementar el método compareTo(T), el cual retorna el resultado de comparar el receptor del mensaje con el parámetro recibido. Este valor se codifica con un entero, el cual presenta la siguiente característica:

= 0: si el objeto receptor es igual al pasado en el argumento.
> 0: si el objeto receptor es mayor que el pasado como parámetro.
< 0: si el objeto receptor es menor que el pasado como parámetro.

La descripción de cada método es la siguiente:

El constructor **ArbolBinariodeBusqueda()** inicializa un árbol binario de búsqueda vacío con la raíz en null.

El constructor **ArbolBinariodeBusqueda(T dato)** inicializa un árbol que tiene como raíz el dato pasado como parámetro y ambos hijos nulos.

El método **esVacio(): boolean** indica si el árbol está vacío (no tiene dato cargado).

Los métodos **get Hijo Izquierdo():ArbolBinariodeBusqueda<T>** y **getHijoDerecho():ArbolBinariodeBusqueda<T>** retornan los árboles hijos que se ubican a la izquierda y derecha del nodo raíz respectivamente. Están indefinidos para un árbol vacío.

El método **incluye (T dato)** retorna un valor booleano indicando si el dato recibido está incluido en el árbol.

El método **buscar (T dato):T** retorna el valor almacenado en el árbol que es igual al dato recibido.

El método **agregar (T dato)** agrega el dato indicado al árbol. En caso de encontrar un elemento igual dentro del árbol, reemplazar el existente por el recibido.

a) Analice la implementación en JAVA de la clase **ArbolBinariodeBusqueda** brindada por la cátedra.

Nota: Tener presente que en ABB no se pueden almacenar cualquier objeto, ya que estos necesitan ser **comparables** para poder ordenarlos dentro de la estructura.

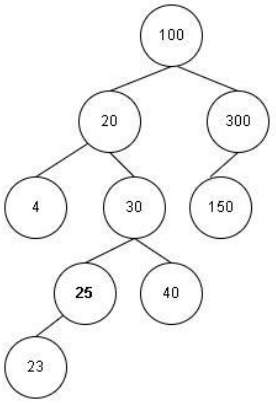
7. Explique con sus palabras (sin implementar) como se realizan las operaciones de inserción y eliminación en un árbol binario de búsqueda.

8. Muestre las transformaciones que sufre un árbol binario de búsqueda al realizar las siguientes operaciones:

a) Insertar las claves 2, 7, 24, 32, 37, 40, 42, 44, 120 en un ABB inicialmente vacío
b) Insertar las claves 120, 44, 42, 40, 37, 32, 24, 7, 2 en un ABB inicialmente vacío
c) Insertar las claves 37, 24, 42, 7, 32, 2, 42, 44, 40, 120 en un ABB inicialmente vacío
d) Al árbol resultante del inciso c) eliminar las claves 120, 2, 37

9. Dado un **ArbolBinariodeBusqueda** de números enteros positivos y mayores a cero, se quiere obtener el camino recorrido hasta llegar a un determinado valor. En caso de ir por la rama izquierda del árbol, el valor a almacenar en el camino será el valor negativo del mismo.

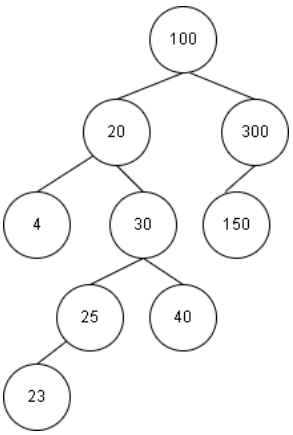
Es decir, en el siguiente ejemplo, si queremos encontrar el valor 25, el resultado será una lista con los valores: 100, -20, 30.



Note que el nodo con valor 20 en el resultado aparece en la lista resultante como -20 debido a que se llegó a este nodo a través del hijo izquierdo del nodo con valor 100. La firma del método a implementar es:

public ListaGenerica<Integer> caminoRecorrido (ArbolBinariodeBusqueda <Integer>)

10.
a) Implemente un método, que dado como parámetros un Árbol Binario de Búsqueda y un valor, devuelve el valor inmediato mayor. Es decir, dado el siguiente árbol:



- Si el valor a buscar es 30, el inmediato mayor es 40
- Si el valor a buscar es 25, el inmediato mayor es 30
- Si el valor a buscar es 300, el inmediato mayor no existe (puede devolver -1)

b) Plantee una solución alternativa.