UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos 2007 **Trabajo práctico Nro. 5**

Árboles Binarios

Objetivos
- Representar árboles binarios e implementar las operaciones de la abstracción
- Realizar distintos tipos de recorridos sobre árboles binarios
- Describir soluciones utilizando árboles binarios

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

ArbolBinario
- NodoBinario raiz
+ArbolBinario() +ArbolBinario(Object dato) -ArbolBinario(NodoBinario nodo) -getRaiz():NodoBinario +getDatoRaiz():Object +getHijoIzquierdo():ArbolBinario +getHijoDerecho():ArbolBinario +agregarHijoIzquierdo(ArbolBinario unHijo) +agregarHijoDerecho(ArbolBinario unHijo) +eliminarHijoIzquierdo() +eliminarHijoDerecho()

NodoBinario
- Object dato - NodoBinario hijoIzquierdo - NodoBinario hijoDerecho
NodoBinario(Object dato) getDato():Object getHijoIzquierdo():NodoBinario getHijoDerecho():NodoBinario setDato(Object dato) setHijoIzquierdo(NodoBinario unHijo) setHijoDerecho(NodoBinario unHijo)

El constructor **ArbolBinario()** inicializa un árbol binario vacío, es decir, la raíz en null.
El constructor **ArbolBinario(Object dato)** inicializa un árbol que tiene como raíz un nodo binario. Este nodo tiene el dato pasado como parámetro y ambos hijos nulos.
El constructor **ArbolBinario (NodoBinario nodo)** inicializa un árbol donde el nodo pasado como parámetro es la raíz. (Notar que **NO** es un método público).
El método **getRaiz():NodoBinario**, retorna el nodo ubicado en la raíz del árbol. (Notar que **NO** es un método público).
El método **getDatoRaiz():Object**, retorna el dato almacenado en el NodoBinario raíz del árbol.
Los métodos **getHijoIzquierdo():ArbolBinario** y **getHijoDerecho():ArbolBinario**, 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 nodo)** para obtener el árbol binario correspondiente.
El método **agregarHijoIzquierdo(ArbolBinario unHijo)** y **agregarHijoDerecho(ArbolBinario 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.

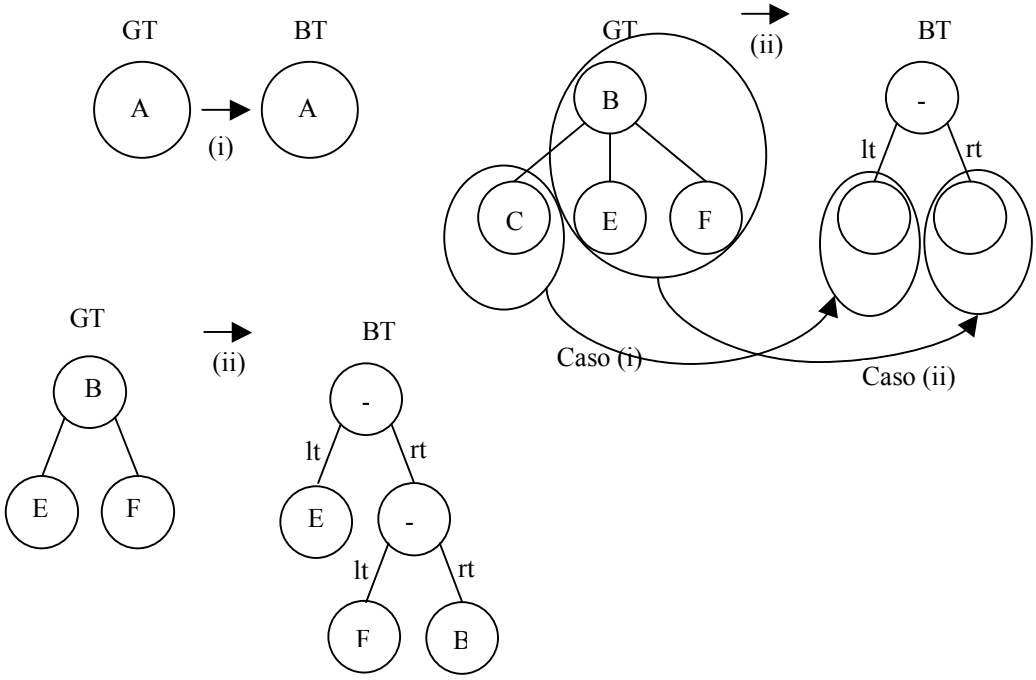
A partir de la especificación anterior, implemente la clase ArbolBinario.

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

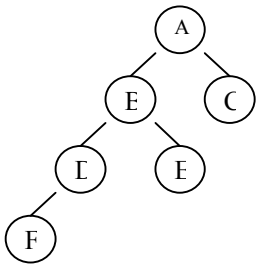
ArbolBinario
- NodoBinario raíz
+ArbolBinario() +ArbolBinario(Object dato) -ArbolBinario (NodoBinario nodo) -getRaiz():NodoBinario +getDatoRaiz():Object +getHijoIzquierdo():ArbolBinario +getHijoDerecho():ArbolBinario +agregarHijoIzquierdo(ArbolBinario unHijo) +agregarHijoDerecho(ArbolBinario unHijo) +eliminarHijoIzquierdo() +eliminarHijoDerecho() **+ArbolBinario (ArbolGeneral unArbol)** **+frontera(): Lista** **+esMenor(ArbolBinario unArbol): boolean** **+zigZag(): boolean** **+lleno(): boolean** **+completo(): boolean** **+espejo(): ArbolBinario** **+colorear()**

NodoBinario
- Object dato - NodoBinario hijoIzquierdo - NodoBinario hijoDerecho
NodoBinario(Object dato) getDato():Object getHijoIzquierdo():NodoBinario getHijoDerecho():NodoBinario setDato(Object dato) setHijoIzquierdo(NodoBinario unHijo) setHijoDerecho(NodoBinario unHijo)

a) **ArbolBinario (ArbolGeneral unArbol).** Inicializa un árbol binario (bt) a partir de un árbol general (gt). La transformación es la siguiente. Cada vértice de *gt* estará representado por una hoja en *bt*. (i) Si *gt* es una hoja entonces *bt* será una hoja con el mismo valor que el de *gt*. (ii) En otro caso, *gt* no es una hoja, entonces *bt* será armado como un vértice raíz (sin contenido), un subárbol izquierdo (*lt)* y un subárbol derecho (*rt)*. *lt* es un árbol binario estricto formado a partir del subárbol más viejo de *gt* (el subárbol más izquierdo de *gt*) y *rt* es un árbol binario estricto, formado a partir de *gt* sin su subárbol más viejo. **Pista:** Tenga presente que deberá cortar el enlace del nodo con el primero de los hijos (primero recupérelo y luego elimínelo).



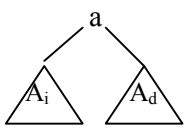
b) **frontera(): Lista**. Se define **frontera** de un árbol binario, a las hojas de un árbol binario recorridos de izquierda a derecha. **Pista:** Recorrer el árbol en preorden, si el nodo es una hoja se agrega a la lista, sino se sigue recorriendo y buscando hojas.



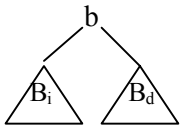
La lista deberá devolver :
F,E,C

c) **esMenor(ArbolBinario unArbol): boolean.** Se define una relación de orden entre árboles binarios de enteros no nulos de la siguiente forma:

$$A < B \begin{cases} a < b; \\ a = b \text{ y } Ai < Bi \\ a = b \text{ y } Ai = Bi \quad \text{y } Ad < Bd \end{cases}$$



árbol A



árbol B

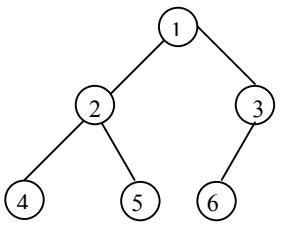
donde a y b son los datos almacenados en los nodos raíces y, Ai, Ad, Bi y Bd son los subárboles izquierdos y derechos. **Pista:** implemente un mensaje privado esIgual(ArbolBinario unArbolBinario): boolean. Este método debería verificar que el árbol es estructuralmente igual y que tiene los mismos valores.

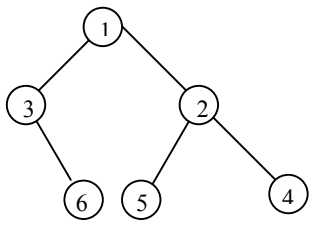
d) **zigZag(): boolean.** Devuelve true si el árbol es degenerado con direcciones alternadas, esto es, si en lugar de ser una lista donde todos los hijos están en el lado izquierdo o derecho, se van alternando.

e) **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). **Pista:** recorra el árbol por niveles. Si encuentra una hoja, todos los demás nodos del mismo nivel deben ser hojas. Si no es así, o si encuentra un nodo que tenga un sólo hijo, el árbol no es lleno.

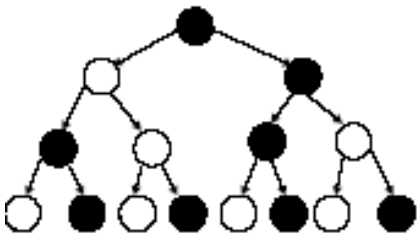
f) **completo(): boolean.** Devuelve true si el árbol es completo. Un árbol binario de altura h es completo si es lleno hasta el nivel (h-1) y el nivel h se completa de izquierda a derecha. **Pista:** Calcule el nivel de la hoja de más a la izquierda (h). Verifique que el árbol es lleno hasta el nivel h-1 con el algoritmo del inciso e. Cuando recorre el nivel h-1, controle que si algún nodo deja de tener hijos, no exista ningún hijo para ningún nodo de más a la derecha.

g) **espejo(): ArbolBinario.** Devuelve el árbol binario espejo del árbol receptor. Por ejemplo:





h) **colorear().** Un árbol binario coloreado es un árbol binario lleno cuyos nodos tienen uno de dos colores posibles: negro o blanco. El color para la raíz del árbol es negro. Y para cada nivel los colores de los nodos se intercalan, como así también al comenzar cada nivel. Por ejemplo un árbol coloreado de altura 3 deberá ser:



Pista: coloree la raíz y encole los hijos para realizar un recorrido por niveles. Encole una marca de fin de nivel. Inicialice una variable con el color a usar. Mientras la cola no se vacía itere. Vaya coloreando los nodos alternando el color y encolando los hijos. Cuando encuentra la marca de fin de nivel, no alterne el color.

Ejercicio 3.
Modelizar e implementar en Java la siguiente situación. Considere un árbol binario no vacío con dos tipos de nodos: nodos **MIN** y nodos **MAX**. Cada nodo tiene un valor entero asociado. Se puede definir el valor de un árbol de estas características de la siguiente manera. Si la raíz es un nodo **MIN**, entonces el valor del árbol es igual al mínimo valor entre: (i) El entero almacenado en la raíz. (ii) El valor correspondiente al subárbol izquierdo, si el mismo no es vacío. (iii) El valor correspondiente al subárbol derecho, si el mismo no es vacío. Si la raíz es un nodo **MAX**, entonces el valor del árbol es igual al máximo valor entre los valores nombrados anteriormente.
Pista: El dato que se guardará en el nodo debe ser un objeto que contenga un valor entero y un valor booleano. El valor false refiere a un nodo min y el valor true a un nodo max. Defina un método **evaluar(): int** en árbol binario. El método debe evaluar() el subárbol izquierdo y el derecho, y debe tomar el valor del nodo. Luego, si es un nodo min, debe retornar el menor de los valores, caso contrario debe retornar el mayor.

Ejercicio 4.
Implemente la clase **ArbolDeExpresion** como subclase de **ArbolBinario.** Incorpore un método de clase llamado **convertirPostfija(String exp): ArbolDeExpresion**, que convierta una expresión aritmética en formato postfijo, en un árbol de expresión. El método recibirá la expresión postfija a convertir (como una cadena de caracteres sin blancos) y devolverá el árbol de expresión correspondiente. Esta expresión es sintácticamente válida, y está compuesta por números de un solo dígito o variables de una sola letra y los operadores binarios +, -, * y /.

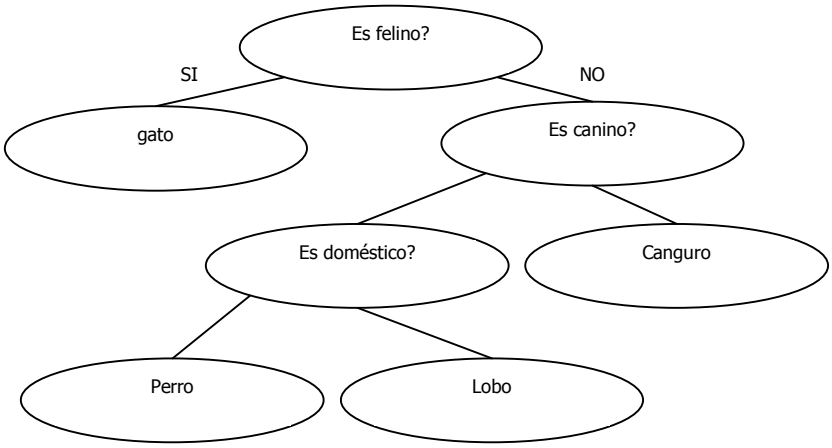
Defina una clase **TestConversion** con su método **main(String[] args)** el cual recibirá un String representando la expresión postfija. Luego utilice el método de clase **convertirPostfija(args) de la clase ArbolDeExpresión,** para obtener el árbol de expresión correspondiente. Finalmente, realice un recorrido postorden imprimiendo cada elemento del árbol (al solo efecto de verificar si se obtuvo el String postfijo original).

Ejercicio 5. - Opcional
Modelizar e implementar en JAVA la siguiente situación. Se quiere hacer que una computadora adivine un nombre de animal que un usuario ha elegido previamente. La computadora debe hacer preguntas a las que el usuario debe responder por si o no. Con estas respuestas la computadora llega a determinar un animal y arriesga una respuesta (haciendo una pregunta del tipo *¿Es un perro*?). Si acierta, el juego termina y la computadora gana. En caso contrario, para que la computadora "aprenda" el usuario debe proveer una pregunta que caracterice al animal en cuestión y que lo diferencie del que la máquina sugirió.
Pista: El juego se debe representar con un árbol binario de decisión: el nodo debe ser una pregunta y la rama izquierda modeliza la elección a seguir si la respuesta fue un **sí**, mientras que la otra modeliza un **no**. En las hojas se encuentran los nombres de los animales. Si la computadora no adivina el animal, se debe modificar el árbol agregando un nodo cuyo nodo raíz tendrá la pregunta que caracterice al animal, su rama del **sí** la respuesta correcta y la rama del **no** el nombre del animal que se arriesgó y resultó incorrecto.

Defina una clase TestJuego con un método main que se encargue de construir un árbol con al menos 3 niveles (que se provee como ejemplo) y luego comience el juego.

Juego
- ArbolBinario preguntas
+ jugar()

TestJuego
- Juego unJuego



Ejemplo: Si la computadora arriesga Canguro, y el hombre había elegido una tortuga, el hombre debe proveer una pregunta que diferencie los dos animales. Podría proveer la pregunta: ¿salta? Que se colocaría en lugar de canguro, tortuga como hijo izquierdo y canguro como derecho.