UNLP. Facultad de Informática

Algoritmos y Estructuras de Datos 2007 **Trabajo Práctico 4**

Árboles Generales

Objetivos
- Modelar un árbol n-ario
- Realizar recorridos e implementar aplicaciones varias sobre el árbol.
- Calcular la complejidad algorítmica de los algoritmos anteriores, en base a la altura o cantidad de nodos.

Ejercicio 1.

Considere la siguiente especificación de la clase **ArbolGeneral** (con la representación de **Lista de Hijos**)

ArbolGeneral
- NodoGeneral raiz
+ArbolGeneral() +ArbolGeneral(Object dato) +ArbolGeneral(Object dato, Lista hijos) -ArbolGeneral(NodoGeneral nodo) -getRaiz():NodoGeneral +getDatoRaiz():Object +getHijos():Lista +agregarHijo(ArbolGeneral unHijo) +eliminarHijo(ArbolGeneral unHijo)

NodoGeneral
- Object dato; - Lista listaHijos; //listaHijos es una Lista de objetos NodoGeneral
NodoGeneral(Object dato) getDato(): Object getHijos(): Lista setDato(Object dato) setHijos(Lista hijos)

Nota: la clase Lista es la utilizada en la práctica 3. Tenga en cuenta que el árbol general también puede implementarse con la representación de **Hijo más izquierdo – Hermano derecho**, sin embargo en la práctica sólo vamos a utilizar la representación **lista de hijos**.

El constructor **ArbolGeneral()** inicializa un árbol vacío, es decir, la raíz en null.

El constructor **ArbolGeneral(Object dato)** inicializa un árbol que tiene como raíz un nodo general, Este nodo tiene el dato pasado como parámetro y una lista vacía.

El constructor **ArbolGeneral(Object dato, Lista hijos)** inicializa un árbol que tiene como raíz un nodo general. Este nodo tiene el dato pasado como parámetro y tiene como hijos una copia de la lista pasada como parámetro. Tenga presente que la Lista pasada como parámetro es una lista de árboles, mientras que la lista que se debe guardar en el nodo general es una lista de nodos generales. Por lo cual, de la lista de árboles debe extraer la raíz (un Nodo General) y guardar solamente este objeto.

El constructor **ArbolGeneral (NodoGeneral nodo)** inicializa un árbol donde el nodo pasado como parámetro es la raíz. Notar que este constructor no es público.

El método **getRaiz():NodoGeneral**, 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 Nodo General raíz del árbol.

El método **getHijos():Lista**, retorna la lista de hijos de la raíz del árbol. Tenga presente que la lista almacenada en el árbol es una lista de nodos generales, mientras que debe devolver una lista de árboles. Para ello, por cada hijo, debe crear un árbol que tenga como raíz el Nodo General hijo.

El método **agregarHijo(ArbolGeneral unHijo)** agrega un hijo a la lista de hijos del árbol. Es decir, se le agrega a la lista de hijos del Nodo General raíz del objeto receptor del mensaje el Nodo General raíz de unHijo.

El método **eliminarHijo(ArbolGeneral unHijo)** elimina unHijo del árbol. Con la misma salvedad indicada en el método anterior.

UNLP. Facultad de Informática

Ejercicio 2.

Agregue a la clase **ArbolGeneral** los métodos en negrita:

ArbolGeneral
+ArbolGeneral() +ArbolGeneral(Object dato) +ArbolGeneral(Object dato, Lista hijos) -ArbolGeneral(NodoGeneral nodo) -getRaiz():NodoGeneral +getDatoRaiz():Object +getHijos():Lista +agregarHijo(ArbolGeneral unHijo) +eliminarHijo(ArbolGeneral unHijo) **+altura():int** **+nivel(Object dato):int** **+ancho():int** **+esAncestro(Object dato1, Object dato2):boolean** **+equals(Object unObjeto):boolean**

a) **altura(): int** devuelve la altura del árbol, es decir, la longitud del camino más largo desde el nodo raíz hasta una hoja. Pista: el mensaje altura debe chequear si el árbol es una sola hoja o no. Si el árbol es una sola hoja, se devuelve 0. Si no, se utiliza el mensaje getHijos() para obtener la lista de hijos (recuerde que devuelve una lista de árboles hijos). Luego, debe iterar por cada uno de los hijos, guardando la máxima altura. A este valor se le debe sumar 1 y retornarlo.

b) **nivel(Object dato):int** devuelve la profundidad o nivel del dato en el árbol. El nivel de un nodo es la longitud del único camino de la raíz al nodo. Pista: si el nodo raíz posee el mismo dato que pasado como parámetro, se retorna 0. En caso contrario, se debe buscar en cuales de los subárboles hijos se encuentra el dato (implemente el mensaje include(Object dato) en la clase Arbol General) y se debe retornar 1 más el nivel que arroje enviar el mensaje nivel() al subárbol que incluye el dato.

c) **ancho():int** La amplitud (ancho) de un árbol se define como la cantidad de nodos que se encuentran en el nivel que posee la mayor cantidad de nodos. Pista: realice un recorrido por niveles. Encole inicialmente la raíz del árbol y luego una marca null (o el número de nivel) para indicar el fin de nivel. Mientras la cola no se vacía, itere. En cada iteración extraiga el tope de la cola, y con la operación getHijos() encole los mismos. Cuando encuentra la marca de fin de nivel cuente si los elementos del nivel es mayor a la máxima cantidad que poseía.

d) **esAncestro(Object dato1, Object dato2):boolean** determina si dato1 es ancestro de dato2, es decir si hay un camino de dato1 hasta dato2. Considere que no se repite el contenido de los nodos. Pista: Implemente un mensaje subArbol(Object dato) que retorna el subárbol que tiene como raíz el nodo general que contiene el dato pasado como parámetro. Tenga presente que el método ancestro debe devolver un valor booleano, y que el mensaje subárbol que se recomienda implementar devuelve árboles.

e) **equals(ArbolGeneral otroArbol):boolean** determina si dos árboles son o no similares. Dos árboles son similares si son estructuralmente iguales (aunque no coincidan los valores). Pista: realice un recorrido por niveles en simultáneo en el objeto receptor y en otroArbol. Si la cantidad de hijos de las raíces de los subárboles no es igual, los árboles ya son distintos. Si son iguales, compare cada para de árboles hijos.

Nota: tenga presente que no importaría qué representación interna utilizaría (lista de hijos o hijo más izquierdo – hermano derecho), estas operaciones deberían estar implementadas a partir de las operaciones especificadas en el ejercicio 1.

UNLP. Facultad de Informática

Ejercicio 3.

Modelizar e implementar en JAVA una estructura de directorio de DOS. Un directorio puede contener archivos y a su vez otros directorios. En la estructura de directorios hay uno destacado que es el directorio raíz. Tanto los archivos como los directorios tienen un nombre, y los archivos también tienen un tamaño. Para este ejercicio codifique métodos para resolver los siguientes problemas:

a.- devolver los archivos de un directorio.
b.- devolver los subdirectorios de un directorio.
c.- retornar el tamaño de un directorio: sólo la suma de los tamaños de los archivos que contiene.
d.- retornar el path (concatenación de los nombres de los directorios) con mayor cantidad de caracteres, comenzando en el directorio raíz.

Pista:

ArbolGeneral
- NodoGeneral raíz
+ArbolGeneral() +ArbolGeneral(Object dato) +ArbolGeneral(Object dato, Lista hijos) -ArbolGeneral(NodoGeneral nodo) -getRaiz():NodoGeneral +getDatoRaiz():Object +getHijos():Lista +agregarHijo(ArbolGeneral unHijo) +eliminarHijo(ArbolGeneral unHijo)

NodoGeneral
- Object dato; - Lista listaHijos; //listaHijos es una Lista de objetos NodoGeneral
NodoGeneral(Object dato) getDato(): Object getHijos(): Lista setDato(Object dato) setHijos(Lista hijos)

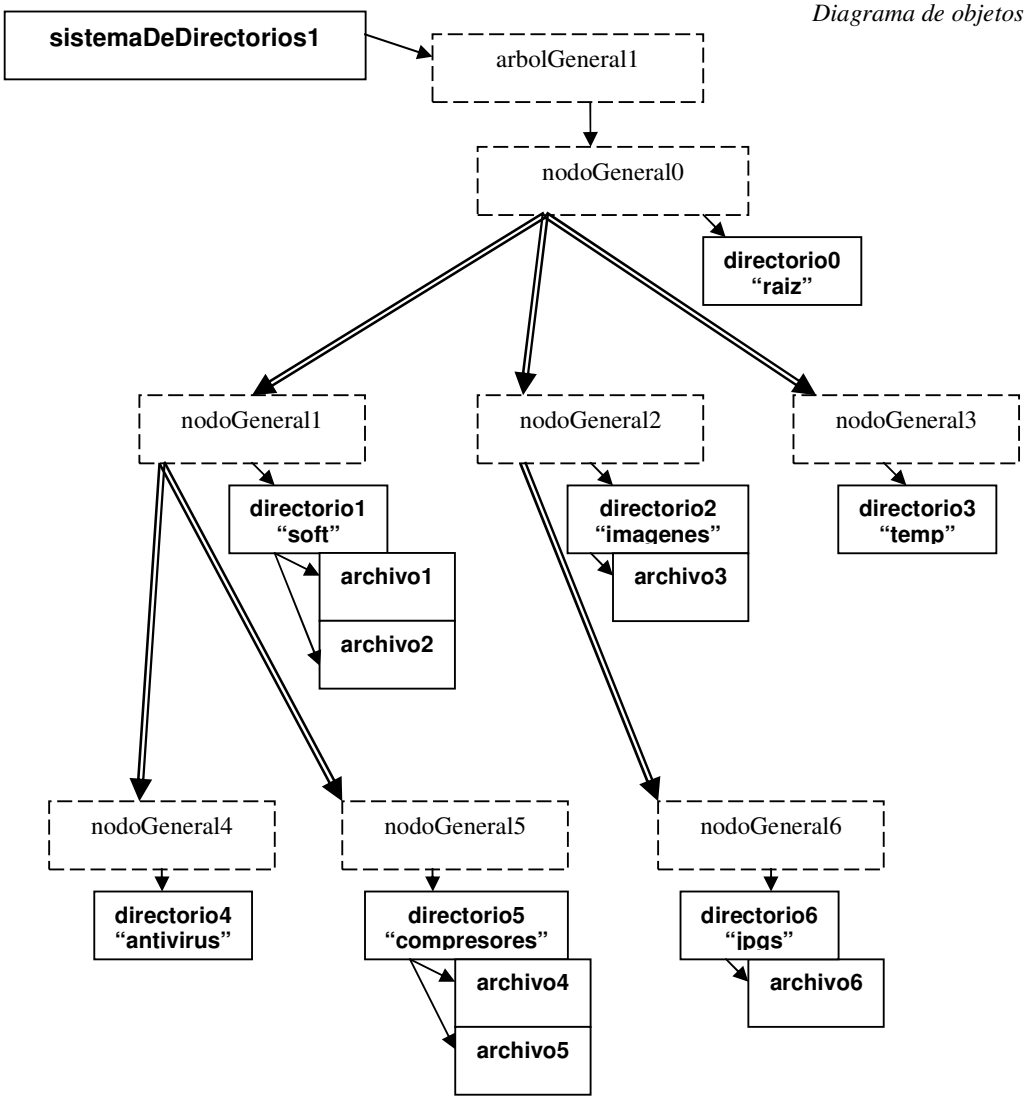
SistemaDeDirectorios
- ArbolGeneral raíz
+archivos(String directorio): Lista +subdirectorios(String directorio): Lista +tamaño(String directorio): int +pathMasLargo(): String

Directorio
- String nombre; - Lista listaDeArchivos
+getNombre(): String +setNombre(String unNombre) +getArchivos(): Lista +setArchivos(Lista archivos)

La estructura de directorios es un árbol general. El dato es un objeto que posee un nombre y una lista de archivos. Esta información debe estar dentro del nodo. Luego, como el directorio es un nodo de un árbol, los hijos del nodo representan los subdirectorios.
a.- debe utilizar buscar el Directorio de forma similar a como busca un subárbol en el ejercicio 2.d. Una vez obtenido el Directorio, debe obtener la lista de archivos.
b.- Idem a, pero debe devolver la lista de hijos (que son los subdirectorios).
c.- debe utilizar el punto 3.a, e iterar la lista de archivos sumando los tamaños.

d.- se deben armar todos los paths posibles utilizando el método getHijos().
e.- Investigue si existe en Java alguna clase que modelice Directorios y archivos.

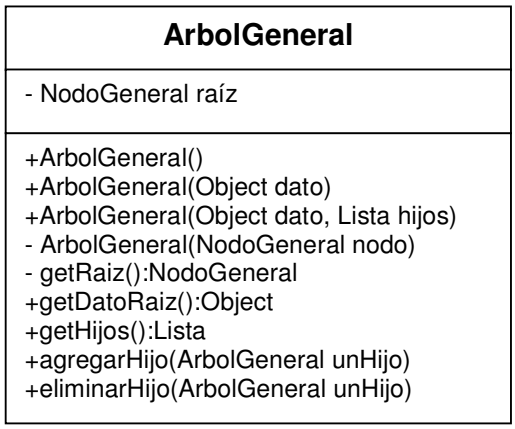
Diagrama de objetos

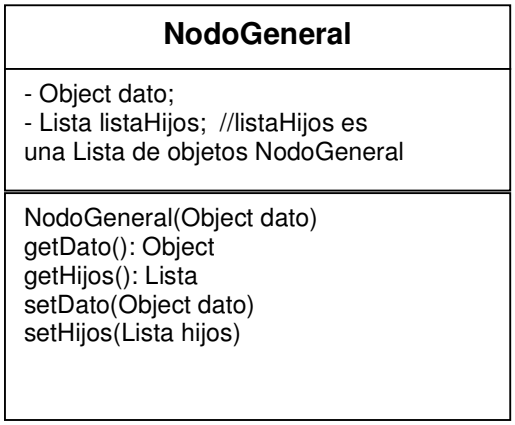


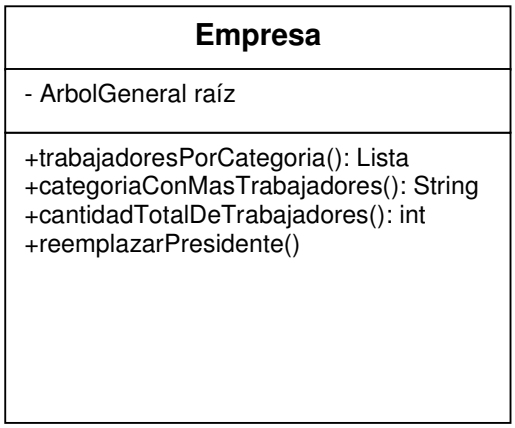
Ejercicio 4.

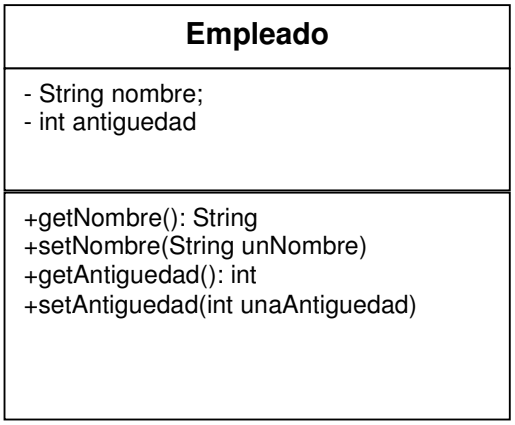
Modelizar e implementar en JAVA la siguiente situación. En una empresa los empleados están categorizados con un número en el rango de 1 a 10, siendo el 1 el presidente, el 2 el vicepresidente, 3 los gerentes y así siguiendo. Los mismos también poseen una antigüedad. Esta información está dispuesta en un árbol general. Para este ejercicio realice métodos que al menos resuelvan los siguientes problemas:
a.- devolver la cantidad de trabajadores por categoría.
b.- determinar la categoría que cuenta con la mayor cantidad de trabajadores.
c.- determinar la cantidad total de trabajadores.
d.- Sea la situación en donde el presidente deja su función, reemplazarlo por la persona más antigua de sus subordinados, quién a su vez es reemplazada de la misma forma.

Pista:









Debe modificar el árbol para que el nodo represente a cada empleado. Luego, los hijos del nodo representan la relación "es subordinado de"

a.- Debe implementar un recorrido por niveles como lo hizo en 2.c. Debe totalizar para cada uno de los niveles.

b.- A partir del punto anterior, se debe quedar con la mayor cantidad.

c.- A partir de a, debe realizar la suma

d.- Debe tomar los hijos de la raíz, y buscar el mayor de los hijos. Sin modificar la estructura, pase el mayor de los hijos a la raíz, y se envía el mensaje al hijo promovido. Cuando el hijo promovido no tenga hijos, se lo debe eliminar.

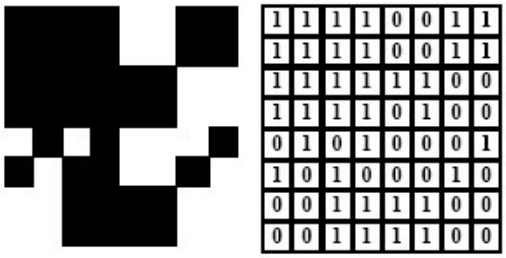
Ejercicio 5.

Implemente una clase en Java llamada **Imagen** que permita crear imágenes en blanco y negro que utilice como representación interna una matriz.

Una forma de comprimir una imagen es transformarla a un árbol 4-ario. El algoritmo es el siguiente. Si toda la matriz tiene un mismo color, se debe definir un nodo con ese color. En caso contrario, se divide la matriz en cuatro partes, se define un nodo con 4 hijos, y cada hijo es la conversión de cada una de las partes de la matriz.

Realice un método de instancia llamado **imagenComprimida()** que devuelva la representación de la imagen en su árbol correspondiente.

Ejemplo:



1	1	1	1	0	0	1	1
1	1	1	1	0	0	1	1
1	1	1	1	1	1	0	0
1	1	1	1	0	1	0	0
0	1	0	1	0	0	0	1
1	0	1	0	0	0	1	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0

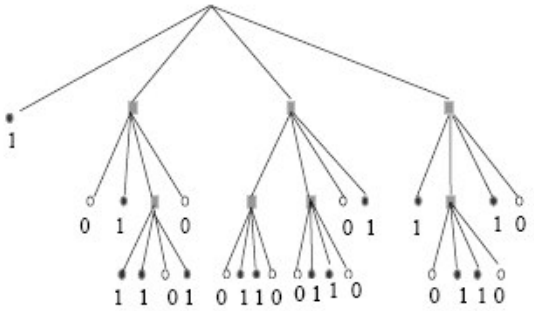


imagen matriz árbol 4-ario

Pista:

ArbolGeneral
- NodoGeneral raíz
+ArbolGeneral() +ArbolGeneral(Object dato) +ArbolGeneral(Object dato, Lista hijos) -ArbolGeneral(NodoGeneral nodo) -getRaiz():NodoGeneral +getDatoRaiz():Object +getHijos():Lista +agregarHijo(ArbolGeneral unHijo) +eliminarHijo(ArbolGeneral unHijo)

NodoGeneral
- Object dato; - Lista listaHijos; //listaHijos es una Lista de objetos NodoGeneral
NodoGeneral(Object dato) getDato(): Object getHijos(): Lista setDato(Object dato) setHijos(Lista hijos)

Imagen
- boolean [] [] imagen
+igualColor(): boolean +color(): boolean +dividirEnSubImagenes(): Lista +imagenComprimida(): ArbolGeneral

Teniendo en cuenta que el color blanco lo representamos con el valor falso y el negro con verdadero, implemente los métodos igualColor(), color() y dividirEnSubimagenes() en la clase Imagen. El método dividirEnSubimagenes() devuelve una Lista con 4 Imágenes. Para comprimir la imagen, debe verificar si la misma posee igual color, en ese caso debe crear y devolver un Arbol General con un único nodo. En cambio, si la imagen no posee el mismo color, debe dividirEnSubImagenes y enviar el mensaje imagenComprimida() a cada una.