**Árboles N-arios y sistemas de archivos**

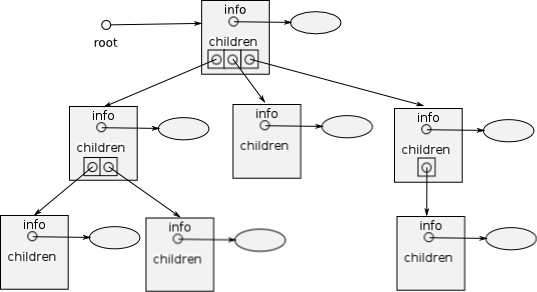
Objetivo

Estudiar otro tipo de árboles más genéricos que los binarios, proponiendo un ejemplo concreto de aplicación: los sistemas de archivos. Se pretende que los alumnos aprendan a generalizar las operaciones que han visto en el caso de árboles binarios para aplicarlas al caso de árboles N-arios.

Árboles N-arios y sistemas de archivos

Hasta ahora hemos trabajado con árboles binarios, pero no son el único tipo de árbol existente. En algunas ocasiones necesitamos árboles más flexibles que nos permitan tener, por cada nodo, un número N de nodos hijo que no tiene por qué ser exactamente dos, y que puede ser diferente para cada nodo. Esta estructura de datos es lo que se conoce como árbol N-ario y se muestra en la figura 1. Cada nodo del árbol contiene una referencia a la información que se quiere almacenar (**info**), y un conjunto de referencias a los nodos hijo (**children**). Para acceder a todos los nodos del árbol tan sólo necesitamos una referencia a su nodo raíz (**root**), tal y como ocurría en el caso de árboles binarios.

Figura 1. Representación gráfica de un árbol N-ario



En este ejercicio vamos a ver un ejemplo en el que se necesitan árboles N-arios: Supongamos que tenemos el siguiente sistema de archivos (también conocido como árbol de directorios):

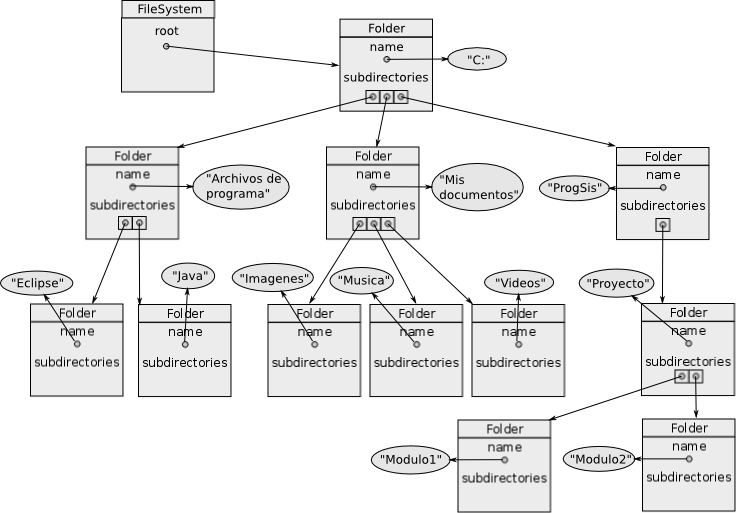
Tal y como su nombre indica, todos los directorios o carpetas (en nuestro caso, para simplificar, vamos a ignorar los archivos) se organizan en forma de árbol: existe un nodo raíz (C:) que contiene varias carpetas, cada una de las cuales contiene a su vez más carpetas, y así sucesivamente. Para crear y manejar este sistema de archivos, vamos a tomar la estructura genérica mostrada en la figura 1 , y la vamos a particularizar para adaptarla al caso que nos ocupa. Los nodos de la imagen serán nuestras carpetas. Cada carpeta está representada mediante un objeto de tipo Folder. Cada uno de estos objetos posee dos atributos:

* name es un atributo de tipo String que almacena el nombre de la carpeta
* subdirectories es un atributo de tipo Vector que almacena los subdirectorios (objetos de tipo Folder) que contiene la carpeta.

Para representar el sistema de archivos, haremos uso de una clase FileSystem que desempeña el papel de árbol, puesto que es la que contiene una referencia al nodo raíz (atributo root de tipo Folder), desde el cual se puede acceder al resto de carpetas (nodos) del sistema de archivos.

La figura 2 representa el sistema de ficheros del ejemplo previo representado mediante los objetos Java que vamos a manejar.

Figura 2. Representación gráfica de un sistema de ficheros modelado con objetos Java



Ejercicio

Para facilitar la realización de este ejercicio, se proporciona parte de la clase Folder, así como la estructura de la clase FileSystem, que se adjuntan a este documento:

* Folder.java
* FileSystem.java

En primer lugar, se va a practicar el uso de objetos de la clase Vector (si no recuerda cómo usarla, entonces consulte la API) y para esto se le pide implementar los siguientes métodos de la clase Folder:

**public Folder addSubdirectory(String folderName) :** añade una nueva carpeta, con nombre folderName , a la colección de subdirectorios y devuelve una referencia a la misma.

**public void printSubdirectories() :** imprime en pantalla el nombre de todos los subdirectorios de la carpeta en el formato: subdirectorio1 subdirectorio2 ... subdirectorioN. Sólo se imprimirán los subdirectorios, sin tener en cuenta los subdirectorios que pudiera tener cada uno de éstos.

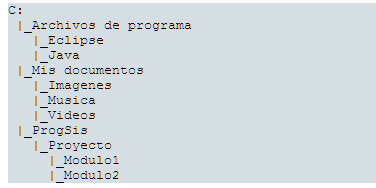
Antes de seguir avanzando asegúrese de que los métodos funcionan correctamente. Para ello, cree una claseFileSystemTest.java para comprobar el funcionamiento.

Una vez tenemos claro cómo funcionan los objetos de tipo Folder y cómo recorrer el objeto de la clase Vector que contiene los subdirectorios (nodos hijo) de cada carpeta, nos centraremos en el sistema de archivos en sí. Para ello se pide implementar los siguientes métodos de la clase FileSystem (además de realizar las pruebas necesarias para comprobar su correcto funcionamiento con la clase FileSystemTest que creaste previamente):

**public Folder searchFolder(Folder root, String folderName):** busca la carpeta cuyo nombre es folderName en todo el árbol. Si la carpeta existe, se devuelve una referencia a la misma, o null en caso contrario. Pista: este método es recursivo ya que, para cada carpeta, hay que recorrer todos sus subdirectorios, los subdirectorios de éstos y así sucesivamente hasta cubrir todo el árbol. Tal vez sea útil utilizar un método auxiliar **private Folder searchFolder(Folder root, String folderName)**, donde **root** será el nodo raíz de cada subárbol sobre el que se vaya a hacer la búsqueda de la carpeta especificada.

**public Folder addNewFolder(String parentFolderName, String newFolderName):** crea una nueva carpeta de nombre newFolderName , y la añade como subdirectorio de la carpeta cuyo nombre es parentFolderName. Si ya existe una carpeta en el sistema de archivos con el nombre newFolderName , o si la carpeta parentFolderName no existe, no se debe hacer nada y se devuelve null . En caso contrario, se crea y añade la nueva carpeta y se devuelve una referencia a la misma.

**public void printFileSystem():** imprime la estructura del sistema de archivos con el siguiente formato:



Pista: este método también es recursivo. Además, el número de espacios en blanco antes de cada nombre tiene mucho que ver con el nivel del árbol en el que se encuentra la carpeta. Tal vez pueda ser útil crear un método **auxiliar private void printFileSystem(Folder root, int level)** con el que se harán las llamadas recursivas.