TECNOLOGÍA DE PROGRAMACIÓN Curso 2015-2016

Guión de la práctica 3

1.- Objetivo de la práctica

- Aprender a aprovechar los mecanismos que proporciona la herencia para maximizar la reutilización de código.
- Aprender a aplicar excepciones jerarquizadas.

Tarea:

Elige un lenguaje orientado a objetos (C++ o Java). Realizarás toda la práctica en dicho lenguaje.

2.- Estructura de datos

Una forma estándar de estructurar ficheros dentro de un computador es mediante un **árbol de directorios**. Cada directorio puede contener varios archivos y a su vez puede contener otros directorios (denominados subdirectorios). El directorio a partir del que parte todo el árbol de directorios se denomina **directorio raíz**. Los nombres de todos los elementos de un directorio son únicos, así que no puede haber nombres repetidos (al igual que en POSIX, se diferencian mayúsculas de minúsculas).

Además de archivos y directorios, los sistemas de ficheros modernos permiten **enlaces**, que corresponden a elementos (archivos o directorios o incluso otros enlaces) que son referenciados desde otro punto del sistema de archivos, posiblemente con otro nombre diferente. Estos enlaces permiten, por ejemplo, acortar rutas largas.

Cuando un usuario está accediendo a algún punto de dicho árbol de directorios, lo hace a partir de un *path* o **ruta**, una secuencia de directorios y subdirectorios desde el directorio raíz hasta llegar al directorio activo. Dicha ruta suele identificarse a partir de la secuencia de nombres de los directorios desde el directorio raíz, separados mediante algún carácter. Para este problema utilizaremos el separador estándar de POSIX (“/”).

Restringimos este problema a las siguientes características:

- Archivos, directorios y enlaces se identifican con un **nombre** (una cadena de texto). El directorio raíz es un caso especial, que se identifica con una cadena vacía. A dicho directorio se accede mediante el carácter separador (“/” en nuestro caso) puesto al inicio de la ruta. Son nombres válidos aquellos que no pueden confundirse con elementos especiales del árbol de directorios (“..”, “.” o cadenas que contengan el carácter separador “/” o espacios).
- Todos los elementos del árbol de directorios tienen un **tamaño** (en bytes).
  - Para un archivo, es una característica inherente que puede cambiar al modificar dicho archivo.
  - El tamaño de un directorio se calcula como la suma del tamaño de todos los elementos que contiene (sean archivos, enlaces simbólicos o subdirectorios).
  - El tamaño de un enlace es el tamaño del elemento al que representa, independientemente de lo que se trate.
- La **ruta activa** es una secuencia de directorios (o enlaces simbólicos a directorios) desde el directorio raíz hasta el subdirectorio correspondiente.
- Para este problema, el **contenido de los ficheros** es irrelevante. Se tienen que poder

crear y borrar ficheros, así como comprobar y modificar su tamaño, pero nada más.

- El sistema de ficheros representado **no es persistente**, sólo está representado en memoria sin acceder a disco en ningún momento. Esto implica que en cuanto finaliza la aplicación toda información del árbol de directorios creado desaparece.
-

Tarea:

Diseña (en dicho lenguaje) las clases necesarias para representar un árbol de directorios, incluyendo ficheros, enlaces simbólicos y la ruta activa. Los nombres de las clases implicadas serán obligatoriamente los siguientes (aunque puedes añadir otras clases si tu solución lo requiere):

- `Directorio` representará un directorio (incluyendo la raíz de un árbol de directorios).
- `Archivo` representará un archivo con su correspondiente tamaño.
- `Enlace` representará un enlace simbólico a un archivo, un directorio u otro enlace.
- `Ruta` representará una ruta sobre un árbol de directorios. El constructor de la ruta deberá recibir como único parámetro un directorio, que será la raíz del árbol de directorios.

3.- Acciones sobre la ruta activa

Todas las acciones que se llevan a cabo sobre el árbol de directorios se realizan a través de la ruta activa. Cada una de dichas acciones se representará mediante un método de la clase que representa la ruta activa. El estado inicial de la ruta activa es el directorio raíz.

Dichas acciones serán representadas por los métodos listados a continuación. El tipo de dato `STR` representa `String` en Java y `std::string` en C++ (cuando se pasa como parámetro deberás poner `const std::string&`)

- `STR pwd()`: Devuelve la ruta completa, con todos los nombres de los directorios desde la raíz hasta el directorio actual concatenados y separados por el separador “/”.
- `void ls()`: Muestra por pantalla los nombres de todos los ficheros, directorios y enlaces contenidos en la ruta actual, cada uno de ellos en una línea diferente, sin ningún dato más.
- `void cd(STR path)`: cambia la ruta a otro directorio (path), pasándole el nombre del directorio al que quiere cambiar. Hay tres casos especiales: “.” se refiere al directorio actual, “..” se refiere al directorio anterior en el árbol de directorios y “/” se refiere a la raíz del árbol de directorios. También se le puede pasar como parámetro una ruta completa (varios directorios separados por “/”).
- `void stat(STR element)`: Muestra por pantalla un número que es el tamaño del archivo, directorio o enlace dentro de la ruta actual identificado por la cadena de texto que se le pasa como parámetro. También se le puede pasar una ruta completa.
- `void vim(STR file, int size)`: Cambia el tamaño de un archivo dentro de la ruta actual (no se le puede pasar como parámetro una ruta completa). Si el archivo no existe dentro de la ruta actual, se crea automáticamente con el nombre y tamaño especificados. Si el archivo referenciado por “file” es en realidad un enlace a un archivo, también cambia su tamaño.

- `void mkdir(STR dir):` Crea un directorio dentro de la ruta actual. No se le puede pasar como parámetro una ruta completa.
- `void ln(STR orig, STR dest):` Crea un enlace simbólico de nombre “dest” a que enlaza el elemento identificado mediante el nombre “orig”. “dest” no puede contener una ruta completa, pero “orig” sí, de tal modo que pueden crearse enlaces simbólicos entre elementos dentro de diferentes posiciones del árbol de directorios.
- `void rm(STR e):` Elimina un elemento dentro de la ruta actual (puede pasársele como parámetro una ruta completa). Si es un archivo es simplemente eliminado. Si es un enlace elimina el enlace pero no lo enlazado. Si es un directorio, elimina el directorio y todo su contenido. Los enlaces a elementos borrados, sin embargo, siguen enlazando al elemento borrado. Así pues, para eliminar completamente un elemento hay que borrar el elemento y todos los enlaces que apuntan a dicho elemento de forma manual.

Tarea:

Sobre la clase que representa la ruta activa (denominada `Ruta`), añade los métodos especificados, con el nombre y los parámetros idénticos a lo especificado en el guión de esta práctica. Además de estos, puedes añadir a cualquier clase todos los atributos y métodos que consideres necesarios para resolver este problema.

4.- Gestión de situaciones excepcionales

Cada una de las acciones del apartado anterior puede generar situaciones excepcionales. Un ejemplo (pero no el único) es el acceso a un directorio inexistente, o el añadir a un directorio un elemento nuevo con el mismo nombre que uno que ya existe en dicho directorio. Dichas situaciones deberán tratarse por medio de una jerarquía de excepciones, que tengan en cuenta todos los posibles casos en los cuales las acciones sobre la ruta activa. Cada excepción podrá tener los atributos y métodos que se consideren necesarios, y el contenido de dicha excepción deberá ser todo lo descriptivo que sea posible.

Tarea:

Diseña una jerarquía de excepciones que gestionen los casos excepcionales del apartado anterior. La clase base de dicha jerarquía se llamara `ExcepcionArbolFicheros`, y cada situación excepcional diferente deberá corresponder a una clase diferente. Cada excepción de la jerarquía deberá sobreescribir el método correspondiente para describir concienzudamente la situación excepcional que ha ocurrido (método `String toString()` en Java o método `const char* what()` en C++)

5.- Pruebas

Toda biblioteca de clases, aunque no debería de ser necesario recordarlo, debería ser probada exhaustivamente, para asegurar de que cumple con la funcionalidad requerida, incluyendo las situaciones excepcionales.

Tarea:

Prueba exhaustivamente las clases que has generado. Como ayuda te proporcionamos un archivo (`Main.java` para Java o `main.cc` para C++) que contiene código que te permitirá probar tu aplicación. Además, te recomendamos que generes tus propios archivos para probar tu estructura de datos y los métodos correspondientes.

Nota importante:

Tu biblioteca de clases deberá compilar (sin ningún error ni aviso de compilación) con el archivo que te proporcionamos, sin necesidad de modificar dicho archivo. Si no lo hace la evaluación de esta práctica resultará en un 0. Esto te obligará a que el nombre de las clases y los correspondientes métodos coincida con lo que te pedimos en este guión.

6.- Consejos

La dificultad de esta práctica depende de decisiones razonablemente simples que se toman al principio (siendo la primera de ellas el lenguaje). Te damos los siguientes consejos para finalizar la práctica sin atascarte en problemas no relacionados con la asignatura:

- Para representar un directorio es probable que necesites una colección o vector de elementos. No utilices los "arrays" básicos de C++ o de Java porque dicha colección crece dinámicamente. Te recomendamos que utilices listas implementadas en las bibliotecas estándar de cada lenguaje (`std::list` en C++ y `LinkedList` en Java).
- Dado que la creación y borrado de archivos, directorios y enlaces en esta práctica fuerza a la reserva y borrado de memoria dinámica, el lenguaje Java te va a facilitar en gran medida la gestión de dicha memoria mientras que en C++ el borrado de dicha memoria tendría que hacerse manualmente, con los riesgos que ello conlleva. Para esta práctica recomendamos (mucho) utilizar punteros inteligentes: `std::shared_ptr`.

Entrega

Deberás entregar todos los archivos de código fuente que hayas necesitado para resolver el problema, excepto el archivo de pruebas que te proporcionamos nosotros (todos menos `Main.java` para Java o `main.cc` para C++). Adicionalmente, para C++, deberás incluir un archivo `Makefile` que se encargue de compilar todos tus archivos `.cc` y generar el ejecutable. No deberás incluir ningún archivo de objeto, ni ejecutable, ni ningún archivo `.class`. Deberás comprimir todos los archivos en un archivo comprimido `.zip`, con el siguiente nombre:

- `practica03_<nip1>_<nip2>.zip` (incluyendo los dos nips de la pareja) si el trabajo se ha hecho por parejas.
- `practica03_<nip>.zip` si el trabajo se ha hecho de forma individual.

Si trabajas en Java asegúrate también que todos los archivos `.java` están en la raíz del archivo comprimido (sin uso de paquetes, que no son necesarios en un proyecto tan pequeño). Al descomprimir el archivo y añadir el archivo de pruebas `Main.java`, deberá poder compilarse y ejecutarse desde su directorio raíz con las siguientes líneas de comandos:

```
javac Main.java
java Main
```

Si no lo hace la evaluación de esta práctica resultará en un 0.

Si trabajas en C++ deberás proporcionar un `Makefile` que se encargue de la compilación. El programa, después de añadir el archivo de pruebas `main.cc`, deberá ejecutarse desde su directorio raíz con las siguientes líneas de comandos:

```
make
./main
```

Si no lo hace la evaluación de esta práctica resultará en un 0.