



## Objetivos

- Aprender a aprovechar los mecanismos que proporciona la herencia para maximizar la reutilización de código.
- Maximizar el polimorfismo y la reutilización de código, eligiendo el mecanismo óptimo de orientación a objetos.
- Aprender a aplicar excepciones jerarquizadas.

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

## 1. Sistemas de ficheros en memoria

Para ciertas aplicaciones es interesante utilizar sistemas de ficheros que no tienen una representación o respaldo en un disco físico, sólo existen en la memoria del computador. Son lo que se denominan sistemas de ficheros en memoria o *ramdisks*.

El objetivo de esta práctica es implementar un sistema de ficheros de tipo *ramdisk*.

Una forma estándar de estructurar el sistema de ficheros dentro de un computador es mediante una estructura de árbol denominada **árbol de directorios**. Cada **directorio** puede contener varios elementos, en particular **ficheros** u otros directorios (denominados **subdirectorios**). A los elementos de un árbol de directorios (sean directorios, ficheros o de otro tipo) se les denomina en general **nodos**. El nodo a partir del que parte todo el árbol de directorios es un directorio y se le denomina **directorio raíz**. Dos nodos dentro del mismo directorio no pueden tener el mismo nombre (se distingue entre mayúsculas y minúsculas).

Además de ficheros y directorios, los sistemas de ficheros modernos pueden almacenar otros tipos de nodos, como por ejemplo **enlaces**, que son un tipo de nodo que referencia a otro (fichero, directorio o incluso otro enlace) almacenado en otro punto del sistema de ficheros, posiblemente con otro nombre diferente. Estos enlaces permiten acceder a un nodo desde otro punto del sistema de ficheros sin copiarlo. Los enlaces referencian directamente otro nodo en el sistema de ficheros, al estilo de los *hard-links* de los sistemas Unix.

Para referenciar de forma textual cualquier nodo dentro de ese árbol de directorios se utiliza su *path* o **ruta**: una secuencia de directorios y subdirectorios desde el directorio raíz hasta llegar al elemento referenciado. Dicha ruta suele especificarse mediante la secuencia de nombres de los directorios desde el directorio raíz, separados mediante un carácter especial (por ejemplo ' / ' en Unix, ' \ ' en Windows o ' : ' en antiguos MacOS).

Para esta práctica concretaremos las siguientes características:

- Utilizaremos el separador definido por el estándar POSIX ( '/ ' ).
- Los nodos se identifican con un nombre (una cadena de texto) sin espacios, y que no contienen el carácter especial de separación ( '/ ' ).
- Existen nombres especiales que no representan un nodo concreto, sino que tienen un significado particular, como ' . . ' (directorio padre) o ' . ' (directorio actual).
- El directorio raíz es un caso especial, no tiene un nombre determinado. Dicho directorio se representa de forma textual (cuando haga falta) mediante el carácter separador ( '/ ' en nuestro caso).
- Todos los nodos del árbol de directorios tienen un tamaño (en bytes):
  - Para un fichero, es una característica inherente que puede cambiar al modificar dicho fichero.
  - El tamaño de un directorio se calcula como la suma del tamaño de todos los elementos que contiene (sean ficheros, enlaces simbólicos o subdirectorios).
  - El tamaño de un enlace es el tamaño del elemento al que referencia, independientemente de su tipo.
- 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 las clases necesarias para representar un árbol de directorios, incluyendo directorios, ficheros y enlaces. Los nombres de las clases implicadas serán obligatoriamente los siguientes (aunque puedes añadir otras clases si tu solución lo requiere):

- **Directorio:** representa un directorio (incluyendo el raíz).
- **Fichero:** representa un fichero con su correspondiente tamaño.
- **Enlace:** representa un enlace a otro fichero, directorio o enlace.

## 2. Ruta activa

A la hora de trabajar con el árbol de ficheros desde un *shell* se define el concepto de *ruta activa*: representa la ruta hasta el directorio dentro del árbol en el que nos encontramos 'situados'.

La ruta se representa como una secuencia de directorios (o enlaces a directorios) a partir de un directorio raíz hasta el subdirectorio correspondiente. La raíz de dicha ruta permite controlar a que ficheros se puede acceder (piensa en entornos tipo *sandbox* o *chroot*).

## Tarea

Define la clase **Ruta**, que 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 sobre el que se define esa ruta.

### 2.1. 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 implementará mediante un método de la clase **Ruta**, que a su vez puede usar métodos definidos en el resto de clases. El estado inicial de la ruta activa es el directorio raíz.

La mayoría de las acciones se aplican al directorio en curso (al que apunta la ruta activa), pero otras acciones requieren acceder a un nodo del árbol (fichero o directorio) distinto, que se especifica de forma textual mediante un parámetro de tipo cadena. Por convenio, los nombres usados para dicho parámetro reflejan su significado:

- **name:** nombre simple, que se supone contenido en el directorio actual.
- **path:** ruta a otro nodo del árbol (fichero, enlace o directorio), que puede ser absoluta o relativa, y que puede contener los nombres especiales ' . ' y ' . . ' en cualquier punto del path.

En todos los casos, el nombre puede hacer referencia a un nodo del tipo requerido (fichero o directorio) de forma directa o a través de uno o varios enlaces, que acaben en un nodo del tipo adecuado.

Dichas acciones se implementan mediante los métodos de la clase **Ruta** listados a continuación (el tipo de dato ficticio 'str' representa String en Java y std::string en C++):

- **str pwd()**  
Devuelve la ruta completa de forma textual, con todos los nombres de los directorios desde la raíz hasta el directorio actual concatenados y separados por el separador ' / '.
- **str ls()**  
Devuelve un listado con el nombre de todos los nodos contenidos en la ruta actual, uno por línea.
- **str du()**  
Devuelve un listado con el nombre y el tamaño de todos los nodos contenidos en la ruta actual, uno por línea.
- **void vi(str name, int size)**  
Edita el fichero de nombre ' name ' (en el directorio actual). Para simular la edición, simplemente se cambia el tamaño del fichero al valor especificado como parámetro. Si el fichero no existe, se debe crear con el nombre y tamaño especificados.
- **void mkdir(str name)**  
Crea un directorio dentro del directorio activo.
- **void cd(str path)**  
Cambia la ruta a otro directorio.  
El ' path ' debe representar un directorio o enlace a un directorio.

- **void ln(str path, str name)**

Crea en el directorio actual un enlace simbólico de nombre 'name' que apunta al elemento identificado mediante el nombre 'path'. El nombre 'name' no puede contener una ruta completa, es un nombre simple, pero 'path' sí, de tal modo que pueden crearse enlaces simbólicos a elementos en otro directorio del árbol.

- **int stat(str path)**

Devuelve el tamaño del nodo que referencia el *path*.

- **void rm(str path)**

Elimina un nodo. Si es un fichero, es simplemente eliminado. Si es un enlace, elimina el enlace pero no el nodo referenciado. Si es un directorio, elimina el directorio y todo su contenido. Si existen enlaces al elemento borrado, ese elemento sigue siendo accesible a través del enlace, pero no a través de su nodo original.

## Tarea

En la clase **Ruta**, que representa la ruta activa, añade los métodos anteriores, 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.

## 3. 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 intentar 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 pueden ocasionar un error.

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, de forma que cuando el programa principal suministrado gestione la excepción, el mensaje de error presentado sea suficientemente informativo.

## Tarea

Diseña una jerarquía de excepciones que gestionen los casos excepcionales del apartado anterior. La clase base de dicha jerarquía se llamará **ExcepcionArbolFicheros** en Java y **arbol\_ficheros\_error** en C++ (siguiendo los convenios de cada lenguaje, verifícalo en el programa principal en cada caso), y cada situación excepcional diferente deberá corresponder a una clase diferente. Cada excepción de la jerarquía deberá encargarse de que el método correspondiente para describir la situación excepcional que ha ocurrido (método **toString()** en Java o método **what()** en C++) devuelve la información adecuada.

## 4. Recomendaciones

La dificultad de esta práctica depende de decisiones razonablemente simples que se toman inicialmente. 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 necesitas que dicha colección crezca dinámicamente. Te recomendamos que utilices alguno de los contenedores de las bibliotecas estándar de cada lenguaje. En concreto, te recomendamos que valores el uso de listas, vectores o diccionarios. Para guardar los elementos de un directorio, deberás utilizar una única colección, que almacene todos los elementos contenidos en el directorio.
- Dado que la creación y borrado de ficheros, directorios y enlaces en esta práctica fuerza a la reserva y borrado de memoria dinámica, la gestión automática de memoria dinámica facilita en gran medida la implementación. El lenguaje Java ofrece de forma directa esa gestión mediante la recolección automática de basura. En C++ el borrado de dicha memoria tendría que hacerse manualmente, por lo que para esta práctica recomendamos (mucho) utilizar los punteros inteligentes (*smart-pointers*) de la biblioteca de C++ (en particular `std::shared_ptr`).

Toda biblioteca de clases debería ser probada exhaustivamente, para asegurar que cumple con la funcionalidad requerida, incluyendo las situaciones excepcionales. Como ayuda te proporcionamos un fichero (`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 ficheros 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 programa principal (`main.cc` o `Main.java`) que te proporcionamos, sin necesidad de modificar dicho fichero. 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.

## Entrega

Deberás entregar todos los ficheros de código fuente que hayas necesitado para resolver el problema, excepto el fichero de pruebas que te proporcionamos nosotros (todos menos `Main.java` para Java o `main.cc` para C++). Adicionalmente, para C++, deberás incluir un fichero `Makefile` que se encargue de compilar todos tus ficheros `.cc` y generar el ejecutable. No deben incluir ficheros objeto (`*.o`) o ejecutables en el caso de C++, ni ficheros de clases compiladas (`*.class`) de Java.

Si la solución está hecha en C++, deberá ser compilable y ejecutable con los ficheros que has entregado y con el `main.cc` que te proporcionamos mediante los siguientes comandos:

```
1 make
2 ./main
```

Si la solución está hecha en Java, deberá ser compilable y ejecutable con los ficheros que has entregado y con el `Main.java` que te proporcionamos mediante los siguientes comandos:

```
1 javac Main.java
2 java Main
```

En caso de no compilar siguiendo estas instrucciones, el resultado de la evaluación de la práctica será de 0. No se deben utilizar paquetes ni librerías ni ninguna infraestructura adicional fuera de las librerías estándar de los propios lenguajes.

Todos los archivos de código fuente solicitados en este guión deberán ser comprimidos en un único archivo ZIP con el siguiente nombre:

- `practica4_<nip>.zip` (donde `<nip>` es el NIP de 6 dígitos del estudiante involucrado) si el trabajo ha sido realizado de forma individual.
- `practica4_<nip1>_<nip2>.zip` (donde `<nip1>` y `<nip2>` son los NIPs de 6 dígitos de los estudiantes involucrados) si el trabajo ha sido realizado en pareja. En este caso **sólo uno** de los dos estudiantes deberá hacer la entrega.

El archivo comprimido a entregar no debe contener ningún fichero aparte de los fuentes que te pedimos: ningún fichero ejecutable o de objeto, ni ningún otro fichero adicional. La entrega se hará en la tarea correspondiente a través de la plataforma Moodle.