Universidad de Santiago de Chile

FACULTAD DE INGENIERÍA INFORMÁTICA

LABORATORIO PARADIGMA ORIENTADO A OBJETOS

JAVA

Paradigmas de Programación
Victor Flores

Autora:
Paloma Zepeda Garrido
paloma.zepeda@usach.cl

Julio 2023

TABLA DE CONTENIDO

***Palabras claves*—Paradigmas, Sistema de Archivos, Programación Orientada a Objetos, Java**

I. Introducción

Durante nuestro recorrido como estudiantes en el campo de la programación, hemos tenido la oportunidad de trabajar con diversos lenguajes de programación que siguen diferentes paradigmas, como el paradigma imperativo, multiparadigma y funcional. En el caso del presente laboratorio, nos hemos adentrado en un lenguaje Orientado a Objetos. Utilizando Java, se ha creado una simulación de un sistema de archivos, explorando temas como la creación, eliminación, búsqueda y otras operaciones relacionadas con archivos, gestionadas a través de comandos de línea.

En el presente informe, se abordará la experiencia en la simulación del sistema de archivos mencionado, incluyendo el análisis, diseño, consideraciones, instrucciones, resultados y conclusiones obtenidas.

II. Descripción del Problema

El sistema de archivos define la forma en que se almacenan los archivos, cómo se organizan en directorios y cómo se accede a ellos. Proporciona métodos para crear, leer, escribir, copiar, mover y eliminar archivos, así como para establecer permisos y atributos.

Se puede definir el sistema de ficheros de un sistema operativo como aquellas estructuras lógicas y sus correspondientes métodos que utiliza el propio sistema para organizar los ficheros en disco.[1]

Este sistema de archivos se basa en los sistemas operativos disponibles en el mercado en la actualidad, como Windows, iOS, Ubuntu, Linux, Android, entre otros. En un sistema de archivos típico, los archivos se organizan en una estructura jerárquica en forma de árbol, donde hay un directorio raíz que contiene subdirectorios y archivos. Los nombres de los archivos y directorios se utilizan para identificarlos y acceder a ellos. Además, proporcionan información adicional, como la fecha de creación, la fecha de modificación, el tamaño, los atributos de seguridad, entre otros.

Las funcionalidades a implementar son las siguientes:

- Autenticación: Poder crear un usuario, logearse y cerrar sesión.
- Crear Unidades físicas.
- Modificación de archivos: Eliminar, añadir, renombrar, buscar, listar.
- Carpetas: Crear carpetas y subcarpetas además de directorios y el poder cambiarlos.
- Registro de actividades: Fechas de creación y modificación tanto de carpetas como archivos
- Papelera de reciclaje: Se crea una papelera de reciclaje, donde se almacenarán archivos eliminados, además de poder vaciarla y/o restaurar elementos.

Para la correcta implementación, haremos uso de seis TDA's:

- System.
- Drive.
- Folder.
- File.
- User.

III. Descripción del Paradigma

Este es un paradigma de programación que utiliza objetos y sus interacciones para diseñar aplicaciones y programas de software. El POO proporciona un enfoque claro y modular para diseñar programas complejos, donde cada componente del programa se trata como un objeto que puede interactuar con otros objetos.

Aspectos importantes del paradigma:

- **Clases:** Una clase es como un plano o una plantilla para crear objetos en el POO. Define un tipo de objeto en términos de los datos que puede contener y las operaciones que puede realizar. Por ejemplo, si tienes una clase Coche, puede tener atributos como color, marca, modelo y métodos como 'acelerar()', 'frenar()', 'cambiarMarcha()'.
- **Objetos:** Los objetos son instancias de una clase. Cuando se crea un objeto de una clase, el objeto contiene su propio conjunto de atributos y tiene la capacidad de realizar cualquier método definido dentro de la clase. En el ejemplo de la clase Coche, podrías tener un objeto miCoche de tipo Coche, que tiene su propio color, marca y modelo, y puede realizar las acciones 'acelerar()', 'frenar()' y 'cambiarMarcha()'.
- **Atributos:** Los atributos, también conocidos como campos, son las variables que pertenecen a una clase. Los atributos contienen el estado o los datos del objeto. En el ejemplo de la clase Coche, color, marca, y modelo son atributos.
- **Métodos:** Los métodos son las acciones o comportamientos que una clase puede realizar. Son similares a las funciones en la programación procedural, pero están asociados a una clase. Siguiendo el ejemplo anterior, 'acelerar()', 'frenar()', 'cambiarMarcha()' serían métodos de la clase Coche.

IV. Análisis del Problema

Al igual que los laboratorios anteriores, se tiene que hacer uso de distintos TDA's para su correcto funcionamiento.

El problema consiste en implementar un sistema de archivos simplificado en Java, donde se puedan crear, modificar y consultar elementos como drives, usuarios y contenido. El sistema se organiza en tres estructuras principales: el TDA System que representa el sistema en su conjunto, el TDA Drive que representa una unidad de almacenamiento, el TDA User que representa un usuario del sistema y el TDA Folder, que representa las rutas y carpetas que se crearán dentro del SO.

- **TDAs:** Se contempla la implementación de los siguientes TDA's:

  1. **TDA System**
  2. **TDA Drive**
  3. **TDA User**
  4. **TDA Folder**

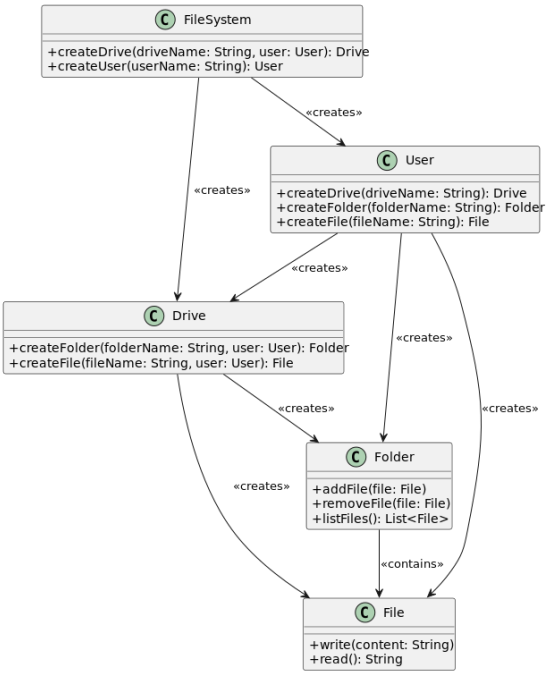

Figura 1: Idea de futura implementación.

Como podemos apreciar en la figura anterior, se contemplan los TDAS mínimos requeridos en el laboratorio.

V. Diseño de la Solución

El proyecto se centró en el diseño e implementación de una versión simplificada de un sistema de archivos, que se asemeja al utilizado en los sistemas operativos modernos. Se utilizó la programación orientada a objetos (POO) para modelar los diversos componentes del sistema de archivos, incluyendo el sistema en sí, las unidades de almacenamiento (drives), los directorios y los usuarios.

En el caso del TDA 'System', se emplean predicados para crear un sistema, modificar los componentes como drives, usuarios y contenido, y realizar consultas sobre ellos. Para el TDA 'Drive', se utilizan funciones para crear una unidad de almacenamiento, así como para modificar el contenido de un drive reemplazando directorios existentes. En el TDA 'User', se emplean predicados para registrar nuevos usuarios y cambiar la unidad actual del usuario.

Recursos empleados:

- **FileSystem:** Esta es la clase principal que representa todo el sistema de archivos. Es responsable de la creación de unidades de almacenamiento, del registro de usuarios y de la gestión de las sesiones de los usuarios.
- **User:** Esta clase representa a los usuarios del sistema. Cada usuario tiene un nombre único y puede iniciar y cerrar sesión en el sistema.
- Drive: Esta clase representa una unidad de almacenamiento en el sistema. Cada unidad tiene una letra única, un nombre y una capacidad, y contiene un conjunto de directorios.
- **Folder:** Esta clase representa un directorio en una unidad de almacenamiento. Cada directorio tiene un nombre, una referencia a su directorio padre y una lista de sus directorios hijos.

Para implementar la funcionalidad de navegación entre directorios, se utilizaron los conceptos de "parent"(padre) y çhildren"(hijos) para representar las relaciones entre los directorios. La variable parentFolder en la clase Folder es una referencia al directorio padre del directorio actual, mientras que childFolders es una lista que contiene todos los directorios hijos del directorio actual.

La interacción del usuario con el sistema se facilita mediante un menú de consola, que permite al usuario seleccionar diferentes operaciones, como la creación de unidades de almacenamiento y directorios, y el inicio y cierre de sesión.

Se aplicó el patrón de diseño de interfaz para permitir una mayor flexibilidad y modularidad en el código. Esto significa que las clases FileSystem, User, Drive y Folder implementan interfaces específicas, lo que permite que la implementación de sus métodos pueda variar sin afectar al resto del sistema.

En términos de recursos, la implementación del proyecto requirió únicamente el lenguaje de programación Java.

Por último, se puso un especial énfasis en la validación de entradas y el manejo de errores para garantizar la robustez y la seguridad de la solución. Esto incluye la comprobación de la unicidad de los nombres de los usuarios y las letras de las unidades de almacenamiento, y la verificación de que el usuario tiene una sesión iniciada antes de permitir ciertas operaciones.

Con todo ya empleado, finalmente el diagrama de clases UML Final quedó así:

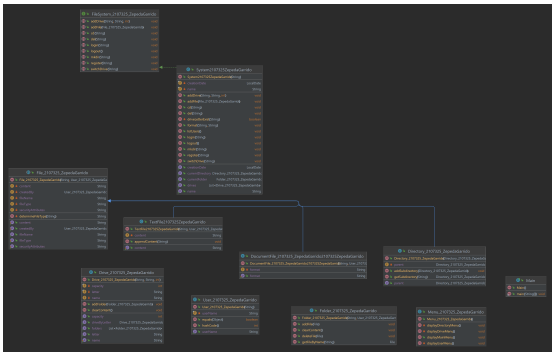

Figura 2: Diagrama UML terminado.

VI. Aspectos de Implementación

1. **Estructura del proyecto:**El proyecto fue organizado en una estructura de paquetes modular, donde cada clase principal, FileSystem, User, Drive y Folder, junto con sus respectivas interfaces, se ubicaron en paquetes separados para mantener la organización del código y la separación de responsabilidades.
2. **Bibliotecas empleadas:** Se utilizaron sólo bibliotecas nativas de Java.
   El proyecto no requirió ninguna biblioteca externa adicional, ya que todas las funciones necesarias se proporcionaron a través del lenguaje Java y su biblioteca estándar. Esto simplificó el proceso de implementación y eliminó la necesidad de gestionar dependencias externas.
3. **Compilador o intérprete usado:**En cuanto al compilador, se utilizó el compilador estándar javac incluido en el JDK (Java Development Kit) para compilar el código fuente de Java en código de bytes, que luego se ejecutó en la JVM (Java Virtual Machine).
   El entorno de desarrollo utilizado fue IntelliJ IDEA, seleccionado por su soporte para Java, sus potentes herramientas de depuración y su capacidad para gestionar y organizar proyectos de gran tamaño.
4. **Disclaimer:** En un principio se estaba complementando el uso de una interfaz gráfica para el presente laboratorio, no obstante me gustaría enfatizar que no he podido seguir inmplementándolo, debido a que estaba teniendo muchos errores, como por ejemplo cuando abría una sección, se abría en una ventana aparte y no desde la misma pestaña, generando que se abrieran múltiples ventanas muchas veces y así con muchos errores, por lo que, si se ve el historial, abré borrado las secciones que lo estaban implementando.

VII. Instrucciones de Uso

Para su correcta implementación, es necesario utilizar un IDE de Java, como se realizó en IntelliJ, es recomentable compilarlo en el mismo programa. Para ejecutarlo, basta con abrir el proyecto, apretar el bóton de ´Build Proyect´, o en su efecto, apretar Ctrl+F9, y cuando el build esté listo y la compilación esté finalizada, basta con correr el proyecto con Mayus+F10.

VIII. Resultados y Autoevaluación

Los resultados fueron los esperados.

Si bien es cierto que no se implementó absolutamente todo, si lo implementado está correcto.

Requerimientos No Funcionales: 1. Autoevaluación: 1 2. Lenguaje: 1 3. Version: 1 4. Documentacion: 1 5. Organización: 1 9. Historial: 1 10. Diagrama de análisis: 1 11. Diagrama de diseño: 1

Requerimientos Funcionales 1.Clases: 1 2: Menú interactivo: 1

```
#### Manipulador de Sistema de Archivos ####
Escoja su opción:
1. Crear un Sistema de Archivos
2. Gestor de Unidades
3. Gestor de Usuarios
4. Gestor de Directorios.
5. Salir
INTRODUZCA SU OPCIÓN: 1
Ingrese el nombre del Sistema de Archivos: NewSystem
Sistema de archivos 'NewSystem' creado exitosamente.


#### Manipulador de Sistema de Archivos ####
Escoja su opción:
1. Crear un Sistema de Archivos
2. Gestor de Unidades
3. Gestor de Usuarios
4. Gestor de Directorios.
5. Salir
INTRODUZCA SU OPCIÓN:
```

Figura 3: Ejemplo de Uso.

```
#### Gestor de Unidades ####
Escoja su opción:
1. Añadir una Unidad al Sistema de Archivos
2. Visualizar Sistema de Archivos
3. Seleccionar unidad de trabajo
4. Volver al menú principal
INTRODUZCA SU OPCIÓN: 2
Nombre del Sistema: NewSystem
Fecha de Creación: 2023-07-17
Unidades:
  Letra: C, Nombre: Acer, Capacidad: 10000
```

Figura 4: Ejemplo de Uso.

1. system: 1 2. add-drive: 1 3. register: 1 4. login: 1 5. logout: 1 6. switch-drive: 1 7. make-directory: 1 8. change-directory: 1 9. add-file:1 10. delete:1 11. remove-directory:0 12. copy:0 13. move:0 14. rename: 15. directory:0 16. format:1 17. encrypt:0 18. decrypt:0 19. grep:0 20. view-trash:0 21. restore:0

Nota final: 5,0

IX. Conclusiones

En resumen, he creado un sistema de archivos que me permite organizar y gestionar mis archivos de manera sencilla. Se pueden crear unidades de almacenamiento, registrar usuarios, crear directorios y agregar archivos a ellos. Además, se puede cambiar de directorio actual para navegar por mi estructura de archivos. El sistema ofrece una interfaz intuitiva en forma de menú para facilitar su uso.

Se ha mejorado un montón respecto a los paradigmas anteriores y se ha hecho menos complicado.

Referencias

[1] R. Juncos, “Sistema de ficheros gnu/linux,” 2008. [Online]. Available:

https://web.archive.org/web/20081214104329/http://observatorio.cnice.mec.es/modules.php?op=modload&name=News&file=article&sid=549