UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

LABORATORIO I

Paradigma Orientado a Objetos Aplicado En La Elaboración De Un Sistema Operativo de Archivos

Paradigmas de Programación - 13310
Profesor Roberto González Ibáñez, PhD.

Byron Caices Lima

17 de julio, 2023

Tabla de Contenidos

1. Introducción 3
2. Descripción Del Problema 3
3. Descripción Del Paradigma Y Conceptos Aplicados 3

Análisis Del Problema 4

4. Diseño De La Solución 5
5. Aspectos De La Implementación 5
6. Instrucciones De Uso 5
7. Resultados Y Autoevaluación 5
8. Conclusión 5
9. Referencias 6
10. Anexos 6

Anexo 1: Estructura de árboles en un sistema de archivos 6

Anexo 2: Diagrama de Análisis UML 7

Anexo 3: Diagrama de Diseño UML 8

1. INTRODUCCIÓN

En el presente informe se abordará el problema propuesto en el enunciado de laboratorio 3 acerca del paradigma orientado a objetos (POO) aplicado en el lenguaje de programación Java. Se revisará una descripción del problema y del paradigma empleado además del diseño de la solución y aspectos de la implementación.

2. DESCRIPCIÓN DEL PROBLEMA

Se presenta como desafío la creación de un sistema operativo de archivos (con orientación al usuario) en donde se tenga un sistema al cual se le pueden añadir unidades o drives y a estas unidades pueden añadirse carpetas y archivos implementando operaciones tales como añadir/formatear unidad, añadir/borrar/renombrar/encriptar carpeta u archivo. Para esto se debe tener en consideración la existencia de una papelera para permitir operaciones como la de restaurar archivo o carpeta. Finalmente se apunta a simular la consola del sistema de archivos introduciendo las operaciones típicas como cd, md, dir, etc.

3. DESCRIPCIÓN DEL PARADIGMA Y CONCEPTOS APLICADOS

El Paradigma de Programación Orientada a Objetos (POO) se basa en la modelización de la realidad, utilizando objetos para representar entidades y procesos. Este enfoque utiliza varios conceptos fundamentales para el diseño y construcción de soluciones de software, que son:

• **Objetos:** Son representaciones activas, o instancias, de una clase en la memoria durante la ejecución de un programa.
• **Clases:** Definen las propiedades (atributos) y acciones (métodos) de un objeto. Actúan como implementaciones de un Tipo de Dato Abstracto (TDA).
• **Atributos:** Propiedades que reflejan el estado del objeto.
• **Métodos:** Acciones que un objeto puede realizar.
• **Constructor:** Método especial para inicializar objetos.
• **Interfaz:** Conjunto de métodos que una clase debe implementar.
• **Composición**: Se refiere a una relación entre clases en la que la existencia del objeto hijo está ligada al objeto padre.
• **Herencia:** Es un procedimiento por el cual una clase (hija) puede heredar atributos y métodos de otra clase (padre).
• **Sobreescritura:** Métodos con igual nombre que realizan acciones distintas según la clase.
• **Sobrecarga:** Métodos con igual nombre pero distintos parámetros para mayor claridad.
• **Diagrama de Lenguaje Unificado de Modelado (UML):** Representación gráfica de la estructura de una solución POO.
• **Diagrama de clase:** Esquema que muestra la interrelación entre clases, sus atributos y métodos.

ANÁLISIS DEL PROBLEMA

Dados los requisitos funcionales específicos que se deben cubrir lo primero que se debe realizar para abordar este problema es la identificación de los TDA necesarios. Notamos que se tienen 5 principales elementos para interactuar en un sistema de archivos: Sistema, Unidades, Items como Carpetas o Archivos y Usuarios.

A su vez, este sistema está compuesto por atributos como un nombre, usuarios registrados, rutas accesibles (paths), unidades, contenido de las unidades y ese contenido posee carpetas y archivos, y las carpetas pueden poseer más carpetas y archivos dentro. Al final, se asemeja a la estructura de árbol del Anexo 1. Sin embargo, implementar una estructura arbórea para la construcción del TDA puede ser un poco engorroso, por lo que nos queda pensar en otra alternativa. Usando los conceptos del POO nos podemos dar cuenta de que el FileSystem es un objeto que posee ciertos atributos como su nombre, usuarios, drives, archivos, carpetas, contenido, rutas, etc. Todos estos atributos previamente mencionados, la mayoría de ellos pueden representarse como otro objeto igualmente, ya que estos poseen estados y comportamientos. Luego, sabemos que el sistema puede tener más de una unidad, ante eso podemos decir que el sistema tiene una lista de drives. A su vez, el sistema en ese caso tendrá una lista de usuarios y de contenido. Contenido el cual corresponde a los tipos de Items que el Sistema puede albergar, para efectos de este proyecto logré identificar dos de estos: File y Folder.

Con esto se puede determinar que para manejar el sistema de archivos basta con trabajar con rutas; añadir una carpeta “Carpeta1” a una unidad “C:” no es más que agregar un path nuevo al sistema “C:/Carpeta1” y además agregar el TDA Carpeta al contenido de la unidad correspondiente para almacenar de alguna forma los TDAs que pueden ser trabajados en el Sistema. Sin embargo, no bastaría solo con agregar la ruta ya que nos faltaría saber, por ejemplo, la metadata de la carpeta o de un archivo como el usuario creador, fecha de creación, fecha de modificación y atributos de seguridad. También quedaría definir qué es borrar un archivo en mi sistema, y se llegó a la conclusión de que borrar una carpeta o archivo no es más que “mover” un item desde el contenido del sistema hacia la papelera del sistema.

Luego considerando que el POO nos permite hacer uso de variables tendremos que trabajar sobre un único Sistema el cual irá recibiendo modificaciones a través de las operaciones planteadas en el enunciado, esto requerirá la implementación de métodos como Selectores y Modificadores que veremos con más detalles en la siguiente sección.

4. Diseño De La Solución

5. Aspectos De La Implementación

Cada función de los requerimientos funcionales se encuentra documentada con su formato javadoc. La solución fue desarrollada en el lenguaje de programación Java JDK 20.0.1 utilizando el IDE IntelliJ IDEA 2022.3.2 en donde el proyecto se integró con Gradle 7.4. No se utilizaron bibliotecas externas.

6. Instrucciones De Uso

7. Resultados Y Autoevaluación

El grado de alcance de los requerimientos funcionales fue parcialmente conseguido casi en su totalidad para las funciones implementadas (12/22 RF). Se cumplieron los resultados esperados a la hora de ejecutar el proyecto.

8. Conclusión

Se lograron los objetivos respecto a desarrollar un sistema operativo de archivos simplificado a través del paradigma orientado a objetos aplicado en una implementación en lenguaje Java. Fue un gran desafío iniciar este proyecto debido a la amplia gama de reglas y conceptos que existen para el paradigma, pero sin embargo fue mucho más abordable que los anteriores paradigmas (funcional y lógico). Se enfrentó el desafío de darle coherencia a la solución no solo a lo que uno puede llegar a diagramar sino que también dicho diagrama debe coincidir con el código pero se logró conseguir las metas propuestas y cumplir con los requerimientos funcionales y no funcionales.

9. REFERENCIAS

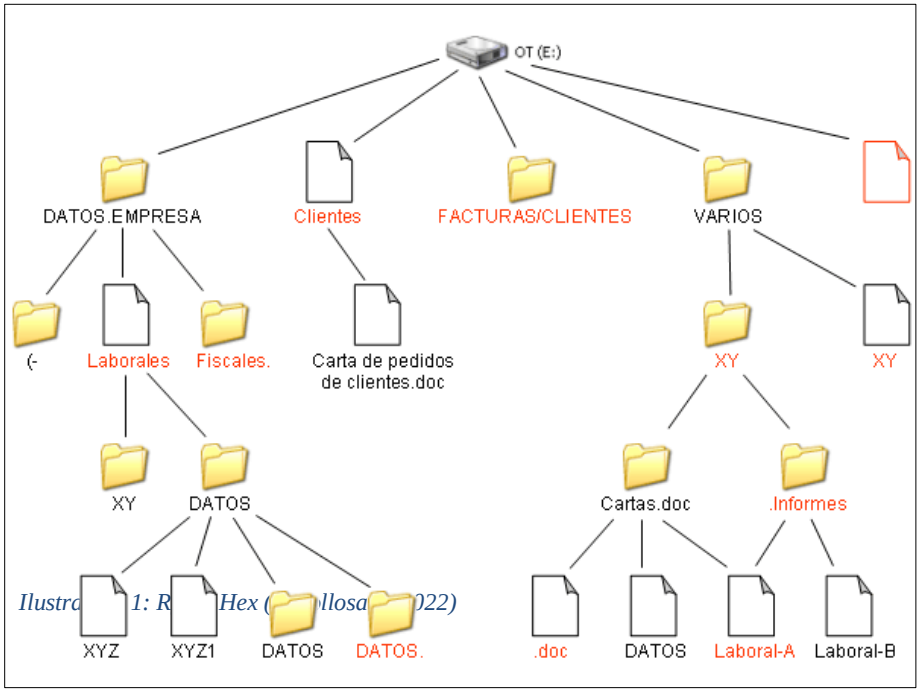
Navarro, S. Universidad San Jorge. (2021, 3 de Junio). *TIPS EN 2 MINUTOS - INFORMÁTICA: Sistema de archivos FAT. Recuperado de https://www.youtube.com/watch?v=bjUhkBBteRM*

Soporte Software. (2014, 1 de Abril). *Sistemas de Archivos.* Recuperado de https://www.youtube.com/watch?v=p2309jhCi9I

Aulapc (s.f). *Archivos, Carpetas, Directorios, Arboles y Pathnames.* Recuperado de http://www.aulapc.es/basico_archivos_carpetas.html

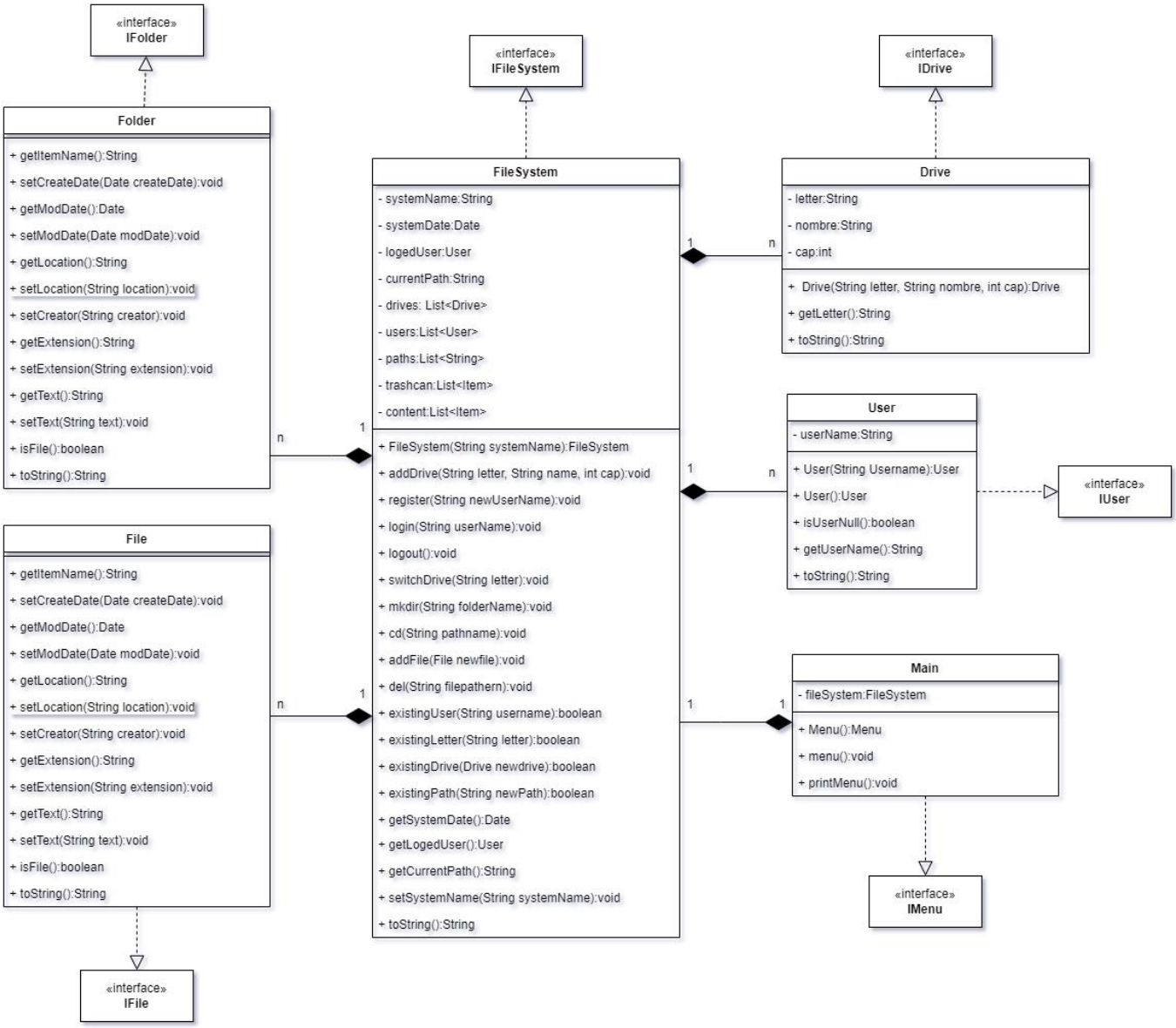
10. ANEXOS

Anexo 1: Estructura de árboles en un sistema de archivos



Ilustra 1: R Hex (llosa 022)

Anexo 2: Diagrama de Análisis UML



Anexo 3: Diagrama de Diseño UML