

UNIVERSIDAD SANTIAGO DE CHILE



# Laboratorio de Paradigmas de Programación

## Laboratorio N° 3: ''Simulación de un Sistema de Archivos en Java o C#''

**Integrantes:**

Claudio Hernández Hernández

**Profesor:**

Edmund Leiva Lobos

**Asignatura:**

Paradigmas de Programación

## 1. Introducción

El presente reporte corresponde al tercer laboratorio del curso de Paradigmas de Programación. En esta ocasión, abordaremos el Paradigma Orientado a Objetos, utilizando el lenguaje de programación Java a través del compilador JDK.

En este informe, abordaremos un problema específico después de una breve descripción de este. A continuación, nos sumergiremos en la descripción del paradigma funcional y un conjunto de conceptos aplicados en este proyecto.

A medida que avanzamos, analizaremos en detalle el problema planteado y diseñaremos una solución acorde con el enfoque funcional. Posteriormente, compartiremos los aspectos más relevantes de la implementación del proyecto y proporcionaremos instrucciones y ejemplos para su uso.

Finalmente, mostraremos los resultados obtenidos y realizaremos una autoevaluación del trabajo realizado. Cerraremos este reporte con una conclusión que destaque los aprendizajes y las conclusiones obtenidas a través del desarrollo de este laboratorio.

## 2. Descripción del Problema

Se requiere crear una simulación de una plataforma que funcione como un sistema de archivos, como, por ejemplo, el Explorador de archivos de Windows. En esta plataforma, se permite la creación de unidades de almacenamiento, y seguido de ello el registro de usuarios, los cuales al identificarse e iniciar sesión correctamente tendrá la capacidad de crear tanto archivos como carpetas, editar el contenido realizado y compartirlo con otros usuarios que también están registrados en el sistema.

Para lograr esto, se deben tener en consideración algunos aspectos fundamentales:

1. Unidades lógicas: Corresponden a partes de almacenamiento que permiten acceder, gestionar y organizar de forma independiente. Cada unidad tiene una letra, un nombre y su capacidad. Estos valores permiten identificar a que unidad nos referimos y la cantidad de espacio que tengamos disponible para su uso.
2. Usuarios: Son a los perfiles que han sido ingresados en el sistema. Estos contienen un nombre de usuario que permite identificarlos. Al momento de iniciar sesión con uno de estos usuarios se le permite el uso de las operaciones disponibles en la plataforma.
3. Archivos: Ficheros creados por el usuario. Estos contienen el nombre del usuario que lo haya creado, fecha tanto de creación como de la última modificación realizada, sus atributos de seguridad y el contenido principal de dicho documento. Estos pueden ser manipulados a través de las operaciones disponibles.
4. Operaciones: Consideradas funciones que permiten realizar distintas acciones en el sistema, como por ejemplo el registro de usuarios, la creación de archivos, remover directorios, entre otros.

## 3. Descripción Paradigma

El paradigma orientado a objetos (POO) es un enfoque de programación que se basa en el concepto de "objetos". En este paradigma, los programas se expresan como representaciones de entidades del mundo real con sus propiedades y comportamientos.

El lenguaje de programación Java se utiliza en este trabajo, aunque no es un lenguaje puramente orientado a objetos, ya que también permite aspectos de programación recursiva o de encapsulamiento.

El paradigma orientado a objetos se basa en la interacción entre atributos y métodos por medio de los mensajes, permitiendo comunicarse gracias a la invocación de métodos.

El POO presenta elementos clave para la construcción de código utilizando este enfoque:

1. Abstracción: Identifica características esenciales de un objeto y los representa en una clase.
2. Encapsulación: Oculta detalles internos de una clase, exponiendo la interfaz pública para interactuar con los objetos de esa clase.
3. Herencia: Crea una nueva clase a partir de una clase existente.
4. Polimorfismo: Un objeto puede tomar muchas formas mediante la capacidad de una clase para utilizar métodos con la misma firma, pero distinto comportamiento.

## 4. Análisis del Problema

Como primer componente fundamental se tiene el sistema, el cual contiene a todos los demás elementos, como usuarios, documentos, directorios, entre otros. Esta plataforma es responsable de registrar y gestionar todos los cambios realizados de manera exitosa a través de operaciones específicas. Es el pilar fundamental que sostiene y organiza todos los datos y acciones que ocurren dentro del sistema.

Considerando esto, se requiere la implementación del sistema en Prolog, el cual puede ser representado de la siguiente manera:

Sistema: Se compone de un string y una fecha. A medida que los demás elementos y/o operaciones sean manipulados, este Sistema se ira actualizando con ello.

También se tienen más elementos importantes en el sistema:

- Unidades lógicas: Corresponden a las unidades de almacenamiento creadas antes del registro de usuarios. Estas listas almacenan una letra que representa su unidad, un nombre y su capacidad de almacenamiento (String x String x int).
- Usuarios registrados: Usuarios que han realizado un registro en el sistema. Se almacena con el nombre del usuario registrado (String).
- Usuario logeado: Pertenece al usuario que ha iniciado sesión en el sistema. Solo contiene al usuario que se ha logeado (String).
- Directorio actual: Como su nombre dice, corresponde a la posición actual del usuario en el sistema. Cada acceso está representado por un string (String x String x ...).
- Carpetas: Directorios que permiten almacenar archivos u otras carpetas dentro de ellas. Estas contienen el nombre de la carpeta, dirección en la que se almacena, usuario que la crea, fecha de creación y de última modificación (String x String x String x Date x Date).
- Archivos: Ficheros almacenados dentro de la unidad lógica o de carpetas. Están representados con el nombre del archivo, el tipo de archivo, su contenido, directorio en el que ha sido creado, usuario que la crea, fecha de creación y de última modificación, y su atributo de seguridad (String x String x String x String x string x Date x Date x String).
- Papelera: Como su nombre dice, representa una papelera de reciclaje. Aquí se almacenan tanto archivos como carpetas que hayan sido eliminadas, para así poder restaurarlas si se desea (list x list x list x ...).

Por último, se tienen las funciones que se pueden realizar en la plataforma:

- system: Construye el sistema.
- systemAddDrive: Añade una unidad con letra única al sistema.
- systemRegister: Registra un usuario único en el sistema.
- systemLogin: Inicia sesión con un usuario existente en el sistema.
- systemLogout: Cierra sesión del usuario
- systemSwitchDrive: Fija la unidad en donde el usuario realizara acciones. Esta solo funciona si hay un usuario logeado.
- systemMkdir: Crea una carpeta con nombre específico dentro de una unidad.
- systemCd: Permite cambiar la ruta en la que se realizaran operaciones.
- systemAddFile: Añade un archivo al sistema en la ruta actual.
- systemDel: Elimina un archivo. El elemento queda en la papelera.
- systemCopy: Copia un archivo en una ruta determinada.
- systemMove: Mueve un archivo a una ruta determinada.
- systemRen: Renombra un archivo o carpeta si el nuevo nombre es único en la ruta.
- systemDir: Lista el contenido de un directorio.
- systemFormat: Formatea una unidad, eliminando todo su contenido.

- `systemEncrypt`: Encripta un archivo o una carpeta con su contenido.
- `systemDecrypt`: Desencripta un archivo o una carpeta con su contenido.
- `systemGrep`: Buscar dentro del contenido de un archivo o de una ruta.
- `systemViewTrash`: Ver el contenido de la papelera.
- `systemRestore`: Restaurar contenido específico de la papelera.

## 5. Diseño de la Solución

Al momento de diseñar la solución, además de utilizar datos nativos de Java, también se deben implementar datos específicos. Estos datos son creados haciendo uso de los llamados TDA, permitiendo la construcción de las operaciones de otros TDAs y para las operaciones mencionadas en el análisis.

Los TDAs más relevantes en este proyecto son:

- `Filesystem`: Representa el sistema de archivos.
- `Drive`: Representa cada disco duro y su información asociada.
- `Folder`: Representa la carpeta que contiene archivos.
- `File`: Representa cada archivo y su información asociada.

Para la modificación y realización de los demás elementos que componen a la plataforma, se tienen las funciones disponibles en el sistema:

### 5.1. OPERACIONES FUNCIONALES

- `systemAddDrive`: Función que realiza una verificación para validar que la unidad sea única, agregando la nueva unidad en caso de que no esté ya almacenada.
- `systemRegister`: Registra al nuevo usuario si este último no está dentro del sistema.
- `systemLogin`: Inicia sesión si el usuario pertenece al sistema.
- `systemLogout`: Si detecta que hay un usuario con una sesión iniciada, la cierra.
- `systemSwitchDrive`: Fija la unidad en la cual se realizarán las acciones, siempre y cuando esta unidad exista en el sistema.
- `systemMkdir`: Función que comprueba si hay un usuario logeado en el sistema, y si la carpeta a agregar es única. Si es así, la agrega al sistema. Si no se cumplen las condiciones, retorna false.
- `systemCd`: Cambia la ruta en la que se realizaran las próximas operaciones, cambiando de directorio o mantenerse en la misma ruta.
- `systemAddFile`: Agrega un archivo a la ruta actual si su nombre es único en dicho nivel.
- `systemDel`: Eliminar uno o varios archivos dependiendo del patrón entregado. También permite borrar carpetas con su contenido.
- `systemCopy`: Realiza una copia de un archivo, recibiendo la ruta de origen del archivo, y la ruta en la que será copiado. también tiene la función de copiar una carpeta o subcarpeta, copiar archivos con una extensión específica o con una inicial específica. En cualquier otro caso, retorna false.
- `systemMove`: Analiza si el archivo a mover ya existe en el destino, y si el archivo a mover existe. En caso de que en el destino ya exista un archivo con su nombre, lo debe sobrescribir.
- `systemRen`: Realiza un cambio de nombre a un archivo específico. Para ello debe verificar que el nombre al cual se quiere cambiar no existe en su mismo nivel.

- `systemDir`: Lista el contenido de un directorio con distintas condiciones, ya que puede listar el contenido del directorio actual, el actual y de sus subdirectorios, listarlas en orden alfabético, entre otros.
- `systemFormat`: Para poder formatear una unidad, primero necesito verificar que esa unidad existe en el sistema. En el caso que si exista, borra todo su contenido, permitiendo además cambiarle su nombre.
- `systemEncrypt`:
- `systemDecrypt`:
- `systemGrep`:
- `systemViewTrash`:
- `systemRestore`:

## 6. Aspectos de Implementación

Este laboratorio utiliza un script de compilación manual en Batch, implementado con el lenguaje de programación Java en su versión `jdk-11.0.20.8-hotspot`.

La estructura del Código es en archivos independientes, donde uno de ellos se considera el archivo principal que implementa una importación de las operaciones y TDAs del código, permitiendo la ejecución de todas las funciones.

En total se tienen 11 archivos, los cuales siguen el siguiente nombre:

- `"NombreTDA-rut-ApellidoPaterno-ApellidoMaterno.pl"`

Exclusivamente, hay dos de los archivos que no contienen el mismo orden que los demás. Estos archivos son el `"main.javaz"` el `compilador.bat`.

## 7. Instrucciones de Uso

Primero se debe comprobar que se tienen todos los archivos en la misma carpeta. Seguido de ello, se debe verificar que el compilador tiene las ubicaciones del JDK, archivos `.java` y almacenamiento de los `.class` correctamente. Con esto ya revisado, se procede a ejecutar el compilador, entregando un mensaje de compilación exitosa en caso de que se haya logrado su función sin fallos, o devolviendo un Error de compilación en caso de que haya ocurrido un problema.

### 7.1. Resultados Esperados

Se busca desarrollar una simulación de un sistema de archivos sin problemas ni ambigüedades. La implementación debe estar libre de errores y utilizar de manera adecuada las estructuras del código.

Cada función debe realizar su tarea correctamente y los Tipos de Datos Abstractos (TDAs) deben tener representaciones adecuadas y bien definidas. En otras palabras, se espera un programa robusto, eficiente y bien organizado que permita un sistema de archivos fluido y sin errores.

## 8. Resultado y Autoevaluación

Los resultados obtenidos fueron los esperados, ya que se crearon alrededor de la mitad los TDAs y de las operaciones funcionales.

Los errores que se pueden esperar son debido a las funciones que no se lograron realizar completamente, debido al ajustado tiempo restante para la entrega del comodín.

Se realizaron múltiples pruebas con distintos ejemplos para comprobar que las funciones no arrojen algún error en la ejecución del Código. Se lograron completar exitosamente 9 de las 22 funciones.

## 8.1. Autoevaluación

Es una lista que se realiza de la siguiente manera:

- 0: No realizado
- 0.25: funciona 25 % de las veces
- 0.5: funciona 50 % de las veces
- 0.75: funciona 75 % de las veces
- 1: funciona 100 % de las veces

## 8.2. Requerimiento No Funcionales

1. Autoevaluacion:1
2. Lenguaje:1
3. Version JDK 11 y Compilador manual:1
4. Interacciones con el programa:1
5. Uso del paradigma: 0.75
6. Prerrequisitos: 0.75
7. Documentacion:0.5
8. Organización del código:0.5
9. Diagrama de análisis:0
10. Diagrama de diseño:1
11. Uso de git:0.5

## 8.3. Requerimiento Funcionales

1. Clases y estructuras que forman el programa: :1
2. Menú interactivo por terminal: :1
3. system:1
4. systemAddDrive:1
5. systemRegister:1
6. systemLogin:1
7. systemLogout:1
8. systemSwitchDrive:1
9. systemMkdir:1
10. systemCd:0
11. systemAddFile:0
12. systemDel:0
13. systemCopy:0
14. systemMove:0

- 15. systemRen:1
- 16. systemDir:0
- 17. systemFormat:0
- 18. systemEncrypt:0
- 19. systemDecrypt:0
- 20. systemGrep:0
- 21. systemViewTrash:0
- 22. systemRestore:0

## 9. Conclusiones del Trabajo

Después de finalizar y concluir el proyecto, se puede afirmar que se lograron cumplir los objetivos principales de manera satisfactoria. Durante el proceso, se adquirió una mejor comprensión del uso de Java y JDK, lo que permitió realizar un gran avance del laboratorio en comparación de la primera entrega. Además, se obtuvo un aprendizaje significativo sobre un nuevo paradigma de programación y se pudo aplicar de manera efectiva, aplicando los conceptos fundamentales aprendidos en clase, como el uso de herencia, encapsulación, entre otros.

La realización del proyecto presentó un desafío significativo debido a la naturaleza de las ideas implementadas. En ocasiones, algunas ideas no funcionaban como se esperaba, lo que requería la revisión y ajuste de enfoques. Lo normal en caso de que ocurriera uno de estos problemas, se optó por realizar cambios cuando era necesario para un correcto funcionamiento. Este enfoque de iterar y adaptarse fue esencial para lograr un proyecto exitoso y funcional.

Nota final esperada: 4.8