

Mini Terminal Unix

17/07/2019

Sandoval Federico.

Programación con Objetos I.

ÍNDICE

Alcance	2
Modelo	3
Progreso	6
Dificultades	8
Conclusiones	9

ALCANCE.

Se desarrolló el trabajo de una Mini Terminal Unix. Se logró implementar todas las partes que el enunciado solicitaba, así como también se desarrollaron características extras al proyecto para darle un sentido más real a su funcionamiento. Entre estas características añadidas se diseñó un objeto que validara los nombres y contraseñas de usuarios junto con los nombres para los archivos. Se diseñó a su vez, la posibilidad de que al acceder a un directorio se pueda volver al directorio anterior además de la característica solicitada por el enunciado de poder regresar al directorio root. Se añadió la clase de Permisos para lograr que un archivo tenga el permiso de lectura, escritura y ejecución, y que los directorios tengan solo permisos de lectura y escritura. Fueron implementados de forma tal que no es necesario ver la implementación de los permisos para saber si están o no habilitados. Por último, se ha diseñado una forma para que los archivos de textos puedan ser ejecutados siempre que tengan cuentas matemáticas, o mensajes que el compilador pueda evaluar, es decir, solo testeamos los casos felices de ejecutar un archivo de texto.

MODELO

Se desarrolló el modelo en base a cuatro objetos principales:

1. MiniTerminalUnix

Es el objeto principal del trabajo ya que se basa en desarrollar un sistema para éste. Posee cuatro colaboradores internos:

- `usuariosRegistrados`: es el colaborador que inicializa siendo una colección en la cual lleva todos los usuarios que se van registrando en la terminal. Comienza con dos usuarios el 'root' y el 'guest'.
- `usuarioActual`: es el colaborador que indica el usuario que se encuentra logueado en la terminal. Inicializa siendo el usuario llamado 'guest'.
- `directorioActual`: es el colaborador que hace referencia al directorio que está viendo el usuario actual de la terminal. Comienza haciendo referencia a el 'directorioRoot'.
- `directorioRoot`: es el colaborador que hace referencia al directorio principal de la terminal. Es un colaborador interno ya que podrían crearse directorios con este nombre dentro de subdirectorios, pero el sistema siempre va a poder identificar cual es el directorio root propio.

Sus principales acciones son: delegar la creación o eliminación de archivos en un directorio; crear y eliminar usuarios; llevar un registro de los usuarios creados; e ir modificando su `directorioActual` conforme se le sea solicitado.

2.Directorío

Es un objeto subclase de 'FileSystem', cuya funcionalidad principal se basa en almacenar archivos e interactuar con la terminal. Posee cinco colaboradores internos:

- nombre: es el colaborador que lleva el nombre por el cual se identifica a dicho objeto.
- owner: es el colaborador que lleva el nombre del usuario que ha creado a este directorio.
- permisosPublicos: es el colaborador que hace referencia a una colección en la cual se encuentran los permisos que posee un directorio.
- directorioPadre: es el colaborador que hace referencia al directorio en donde se crea este nuevo directorio (subdirectorio).
- contenido: es el colaborador que hace referencia a una colección de todos los archivos que fueron creados dentro del directorio.

Sus acciones principales son crear y eliminar archivos de su contenido, habilitar o deshabilitar sus permisos, y generar una lista con la información de los archivos que están en su contenido.

3.ArchivoDeTexto

Es un objeto cuya superclase es 'FileSystem', por lo que comparte protocolo con el objeto 'Directorio', como los colaboradores internos de: nombre, owner y permisosPublicos. Pero a su vez, tiene otros comportamientos y un colaborador que no comparte, como lo es:

- contenidoTextual: es el colaborador interno que hace referencia al contenido del archivo. Ya sean oraciones, cuentas, expresiones, preguntas, órdenes, en otras palabras, cualquier texto escrito posible.

Entre sus acciones principales se destaca el poder escribir y sobrescribir en su contenido textual y el poder ejecutar el contenidoTextual en caso de que sean expresiones que puedan ser evaluadas por el lenguaje (Smalltalks-Cuis).

4. Usuario

Es una clase la cual identifica a los usuarios de la Mini Terminal Unix. Sus acciones principales son basadas en el comportamiento de la miniTerminal, ya que este último accionará dependiendo del usuario que se encuentre logueado en él. Por lo que no posee acciones en sí, sino validaciones de algo, como lo es validar si es o no un usuario root, si puede o no crear usuarios, o bien, si es un usuario de un nombre determinado. Posee cuatro colaboradores internos:


- nombreDeUsuario: es el nombre con el cual se identifica a un usuario.
- contraseña: es la cual se relaciona con el nombreDeUsuario para poder iniciar sesión en la terminal y poder crear un usuario.
- tipoDeUsuario: hace referencia al tipo de usuario que se creó. Hay dos tipos en este sistema: el 'root' y el 'comun'.

PROGRESO

El trabajo fue realizado con Test-Driven Development(TDD). Se comenzó con el diseño básico de los Usuarios. Se los desarrolló con TDD y surgieron dos subclases, pero al transcurrir el trabajo se notó que poseían comportamientos muy similares, por lo que luego con refactors y algunos cambios en la implementación de ciertos mensajes se llegó a que solo quedara la clase Usuario con un colaborador interno que definiera el tipo de usuario que se instancia. Dicha instanciación se realiza a través de los constructores, donde tiene mensajes para instanciar un usuario común (todos los usuarios que se registran en la terminal) o un usuario root(el usuario que se instancia solo una vez para que la terminal posea un root que pueda registrar usuarios).

Luego de los usuarios se comenzó a diseñar el sistema de archivos de la terminal. Con el TDD, surgieron dos subclases principales: 'ArchivoDeTexto' y 'Directorio', ambos compartiendo protocolo establecido en la clase 'FileSystem'. Al comienzo solo se pudo diseñar sus funciones principales de cada subclase, para ArchivoDeTexto se implementaron mensajes para que se puedan leer, escribir y sobrescribir; y para Directorio se implementaron las funciones de que se puedan crear y guardar archivos en uno y que pueda encontrar un archivo general, un archivo de texto o un directorio determinado. En 'FileSystem' se crearon mensajes para que cada archivo del sistema sepa cambiar su nombre y dar su información, lo que incluye su nombre y el nombre de su creador (más adelante se agregó la función de mostrar los permisos que posee).

Ya con los usuarios y el sistema de archivos con unos diseños establecidos, se optó por implementar la clase MiniTerminalUnix. Aquí fue donde se desarrolló la parte fuerte del trabajo ya que este objeto integraba tanto a los usuarios y los archivos para dar la idea de que es una verdadera terminal. Se comenzó diseñando el cómo es que iba a inicializarse dicha terminal, ya que fue algo complicado de decidir: se tenía que elegir con qué usuario logueado comenzaba, con cuántos y qué usuarios registrados y cómo partir de un directorio principal y que siempre sepa cuál era dicho directorio. Luego se decidió que comenzaría con dos usuarios llamados 'guest' y 'root', que el usuario logueado sería el 'guest', y que siempre recuerde que su directorio root era uno determinado, que por más nombres repetidos que se encuentren en los subdirectorios, siempre sepa cual es el verdadero directorio principal del sistema, que sería donde se comienza cada vez que se inicia una nueva sesión y por lo tanto, cuando se inicia una nueva terminal. Luego se diseñó la creación y eliminación de usuarios: se realizaron varios estados para, primero, corroborar que el usuario que crea a otro sea uno de tipo 'root', luego, en el caso de creación, que el usuario que se crea no exista otro con ese nombre en la terminal, y en el caso de eliminación, la misma verificación de que sea 'root', y que el usuario que se desee eliminar exista efectivamente en el registro que lleva la MiniTerminalUnix en su



colaborador interno de usuarios Registrados. Más adelante, a modo de enriquecer el trabajo, se diseñó un objeto 'CorroboradorDeCaracteres', cuya función, básicamente, es que los nombres y contraseñas de un usuario que se quiera crear cumpla con ciertos requisitos que se establecieron en dicho objeto. Luego, se diseñó la creación de archivos, ya sean textuales o directorios, de la siguiente manera: también se han realizado estados por motivos similares a los de crear usuarios, como que no exista otro archivo con el nombre que se quiere crear, que el que crea algo en dicho directorio sea el usuario root o el usuario que haya creado el directorio donde se quiere crear dicho archivo. Una vez diseñada la creación de archivos, se diseñó la interacción que se establece entre un archivo y un usuario a través de la terminal, por lo que surgieron la creación de los mensajes para crear y eliminar archivos en un directorio, buscar cierto archivo de texto para poder ver su contenido o acceder a un subdirectorio del que se encuentra la terminal, poder conocer la información de los archivos que contiene el directorio actual de la terminal, poder escribir o sobrescribir un archivo de texto y cambiar los nombres de un archivo determinado. También se creó un estado para poder iniciar sesión en la terminal, el cual verifica que exista el usuario con la contraseña ingresada. Al mismo tiempo, añadimos la característica de poder cerrar sesión, lo cual conlleva a que el usuarioActual vuelva a ser con el que comienza la terminal, el usuario 'guest'.

Una vez que ya se tenía un modelo con una buena base, se optó por modelar los permisos de los archivos. Por lo que ahora, cada acción que se quería realizar a un archivo desde una terminal con cierto usuario logueado, primero se verifica a través de un estado el comportamiento que debía tomar el objeto dependiendo de que, si bien era el usuarioRoot (el cual posee todos los permisos sobre todos los archivos del sistema), el usuario owner (el cual posee todos los permisos sobre los archivos que crea), o bien, en caso de ser un usuario distinto a los mencionados, se chequea si dicho archivo sobre el cual se quiere accionar posee permisos para realizar la acción deseada. De este modo, los únicos usuarios que pueden setear permisos a un archivo son el root del sistema o el owner del archivo, por lo que ahí también se implementa un estado para que nadie más que estos usuarios puedan setear permisos a un archivo determinado.

Así se logró modelar la mayoría de los ítems solicitados, por lo que se procedió a diseñar que la terminal pueda generar un stream con la información de los archivos que contiene el directorio en el cual se encuentra parada la terminal.

Luego se diseñó un objeto llamado 'ErroresDeTerminal', el cual recibe mensajes para levantar todas las excepciones que hay en el sistema, esto fue hecho para que los mensajes de errores sean delegados a este objeto.

Finalmente se realizaron refactors en los tests para darles mayor legibilidad, se realizaron extract methods para que los métodos de ciertos mensajes sean más fáciles de comprender y se asignaron mejores nombres a clases y mensajes que no eran tan felices.

DIFICULTADES

En el trabajo se presentaron las siguientes dificultades:

- identificar los estados y el cómo hacer el handling;
- también hubo conflictos sobre el cómo debería interactuar la terminal con los usuarios y directorios;
- el cómo crear un nuevo archivo en un directorio determinado;
- ha sido difícil diseñar los permisos, hubo varias formas, pero luego decidí por crear el objeto permiso y que pueda tener un colaborador que identifique su tipo de permiso (lectura, escritura o ejecución);
- el cómo asignarle los permisos a un directorio y luego a un archivo de texto;
- fue difícil testear los casos del stream,
- y el entender cómo funcionaban los streams y guardarlos en un archivo real.

Luego se hizo sencillo:

- Establecer jerarquía de clases;
- Diseñar estados;
- Delegar responsabilidades en los métodos de los mensajes;
- El diseño de la terminal,
- y añadir objetos extra para enriquecer el trabajo;

CONCLUSIONES

Utilización del método TDD.

Gracias a este nuevo método aprehendido para programar y diseñar un código, identificamos que es mucho más útil y eficaz que otros métodos que nos han enseñado anteriormente, ya que gracias al TDD, pudimos testear pequeñas porciones de código, diseñar hasta cierto punto y dejar de trabajar sobre el código sabiendo todo lo que andaba y lo que fallaba, así, comenzamos armandolo desde lo más fácil y esencial, y fuimos integrando objetos cada vez más complejos o complementarios para el trabajo casi sin esfuerzo, ya que no teníamos que parar para realizar suposiciones, simplemente se nos ocurría una idea, la escribíamos y la testeabamos en el momento, quitando la presión de suponer cosas que no sabríamos si pasarían, lo que nos llevó a construir un programa completamente entero a base de probar ideas, equivocarse, decidir el mejor camino, e intercambiar opiniones, lo cual fue muy grato para el trabajo en equipo.

Utilización del Debugger.

Este ha sido de gran ayuda para los momentos en los cuales no podíamos encontrar los errores en los tests. Al principio no costaba adaptarnos a él, ya que no podíamos comprender específicamente la idea de tenerlo, pero luego, cuando comenzábamos a trabarnos en implementaciones o en diseño de mensajes, éste nos mostraba claramente en qué mensaje para qué objeto y en qué momento fallaba lo que hacíamos, lo que resultó muy útil para desarrollar con mayor facilidad el trabajo.

Concluimos que es una de las mejores herramientas brindada por el lenguaje.

Aplicación de conceptos esenciales.

- Polimorfismo: Pudimos utilizar varios mensajes polimórficos entre varias clases para determinar el comportamiento que cada una debería tomar al momento de recibir un mensaje, ya que los objetos tomarán diferentes comportamientos a pesar de que tomen el mismo mensaje.
- Estados: Es el concepto más importante que incorporamos para desarrollar el trabajo, ya que en muchas ocasiones se precisó distintos comportamientos para un mismo mensaje, y optamos por los estados para desarrollar esta idea en lugar de implementar condicionales.
- Protocolos: Establecimos varios protocolos en las clases para establecer comportamientos determinados para darles una identidad única a cada clase del trabajo.
- Jerarquía de clases: este concepto nos permitió establecer ciertos protocolos a seguir para objetos con una misma superclase (como lo son el directorio y el archivoDeTexto), y a su vez, junto con el polimorfismo, que adopten comportamientos distintos con algunos mensajes y el mismo comportamiento con otros gracias a que al ser subclases del mismo objeto (fileSystem), heredan los mensajes establecidos en esta superclase como su protocolo.
- Colecciones: Hemos utilizado las colecciones en todos los objetos principales de nuestro trabajo, especialmente para que en la terminal se pudiera integrar un sistema que contemplara los usuarios y los directorios, y pudiera interactuar entre sí con total facilidad.
- Contrato de mensajes: Gracias a establecer buenos nombres para los colaboradores externos que acompañaban a los mensajes de tipo keyword, los cuales indican el tipo de colaborador que esperan, y el uso de buenos nombres para los mensajes, estimamos haber logrado un programa feliz y legible , y así, finalmente poder comunicar la idea y el comportamiento de cada objeto con los mensajes establecidos en su clase.