Laboratorio 3: Paradigma Orientado A Objetos

Rodrigo Mardones Aguilar

8/10/2020

Indice

Introducción

Los Paradigmas de programación son modelos o patrones de trabajo, los cuales nos permiten resolver problemas abordando de distintas manera la ejecución de la solución.

Es preciso señalar que indiferente de los lenguajes nuevos, modernos y sus herramnientas propias asociadas, siempre se regirán por al menos uno o más paradigmas de programación. Pues son estos quienes nos permiten analizar de buena manera

Objectivo General

Para este laboratorio n°3 se pretende construir una pieza de software que permita simular un "controlador de versiones" tipo git, utilizando como requerimiento base el paradigma orientado a objetos y las caracteristicas que rigen a este paradigma.

Objectivos Específicos

Para abordar el problema se plantean lo siguientes objectivos especificos

- Diseño de la solución.
- Aspectos de la implementación.
- Instrucciones de uso.
- Resultados y Autoevalución.
- Conclusiones.

Descripción del problema

La necesidad de mantener y producir software escalable, ordenado y mantenible durante el tiempo es una cuestión importante dentro de la industria. Poder llevar todos los cambios asociados a un proyecto, trabajado por un grupo importante de personas y todo lo que conlleva el manejo de errores es parte de lo que resuelven los software denominados "controladores de versiones".

Para el problema en cuestión se requiere de una pieza de software que permita realizar las siguientes acciones dentro de un directorio de trabajo, independiente del trabajo realizado dentro de este:

- Poder guardar cambios de un proyecto en cuestión, asociados a un espacio de trabajo y a un usuario.
- Poder revisar, listar y mostrar los cambios guardados de un proyecto asociado.
- Poder enviar y traer los cambios que se tienen guardados a un repositorio externo.
- Poder mostrar las zonas y sus diferentes estados dependiendo de la información contenida en estos.

Descripción del paradigma

El "paradigma orientado a objetos" es una paradigma de programación cuya unidad de trabajo son los "objetos", medidas de representación y modelado del mundo real. Estos objetos, pueden relacionarse con otros objetos mediante el uso de mensajes que puedan interpretar.

La definición de un objeto está dada por las "Clases" que son moldes o estructuras definidas para la construcción de objetos semejantes a la clase planteada. Las clases a su vez tienen atributos y metodos. Los atributos son las caracteristicas(variables) que componen a la clase y los metodos(funciones) son los comportamientos definidos para esta misma.

Caracteristicas definidas del paradigma

Como ya mencionamos, su unidad principal son los "objetos", y la construcción o creación de estos se asocia al concepto de "Clase". Otras caracteristicas importantes a destacar son las siguientes:

- Herencia: Es la capacidad de "heredad" comportamientos y caracteristicas de una clase "padre" a otra "hija", ayudando así a la reutilizacion de codigo y el entendimiento logico.
- Polimorfismo: Esta propiedad relacionada con la herencia, hace referencia al comportamiento de una clase hija, la cual puede comportarse tanto como por su definicion y como por la definición de la clase padre que hereda.
- Sobrecarga: La sobrecarga permite redifinir métodos que experimenten comportamientos distintos dependiendo también de los parametros que son enviados a estos. así podemos tener más de una defincion para el mismo método pero con respuestas distintas.
- Sobreescritura: La sobreescritura se entiene como la redefinición completa de un metodo heredado del padre, para así ajustarlo al comportamiento deseado por la clase hija.
- Clase Abstracta: se entiende por "clase abstracta" la definición de una clase en la que no se puede hacer una instancia directa. Otra definión más entendible es la implementación parcial de un TDA(tipo de dato abstracto), pues permite definir de manera parcial el comportamiento deseado de un representación.
- Interfaces: Las interfaces a su vez son representaciones completas de un TDA, definiciones características y comportamientos ordenados.

Analisis del problema

Para el problema en cuestion se han dispuesto de distintas zonas de trabajo las cuales consituyen un flujo ordenado y unidireccional. estás zonas corresponden a las siguientes:

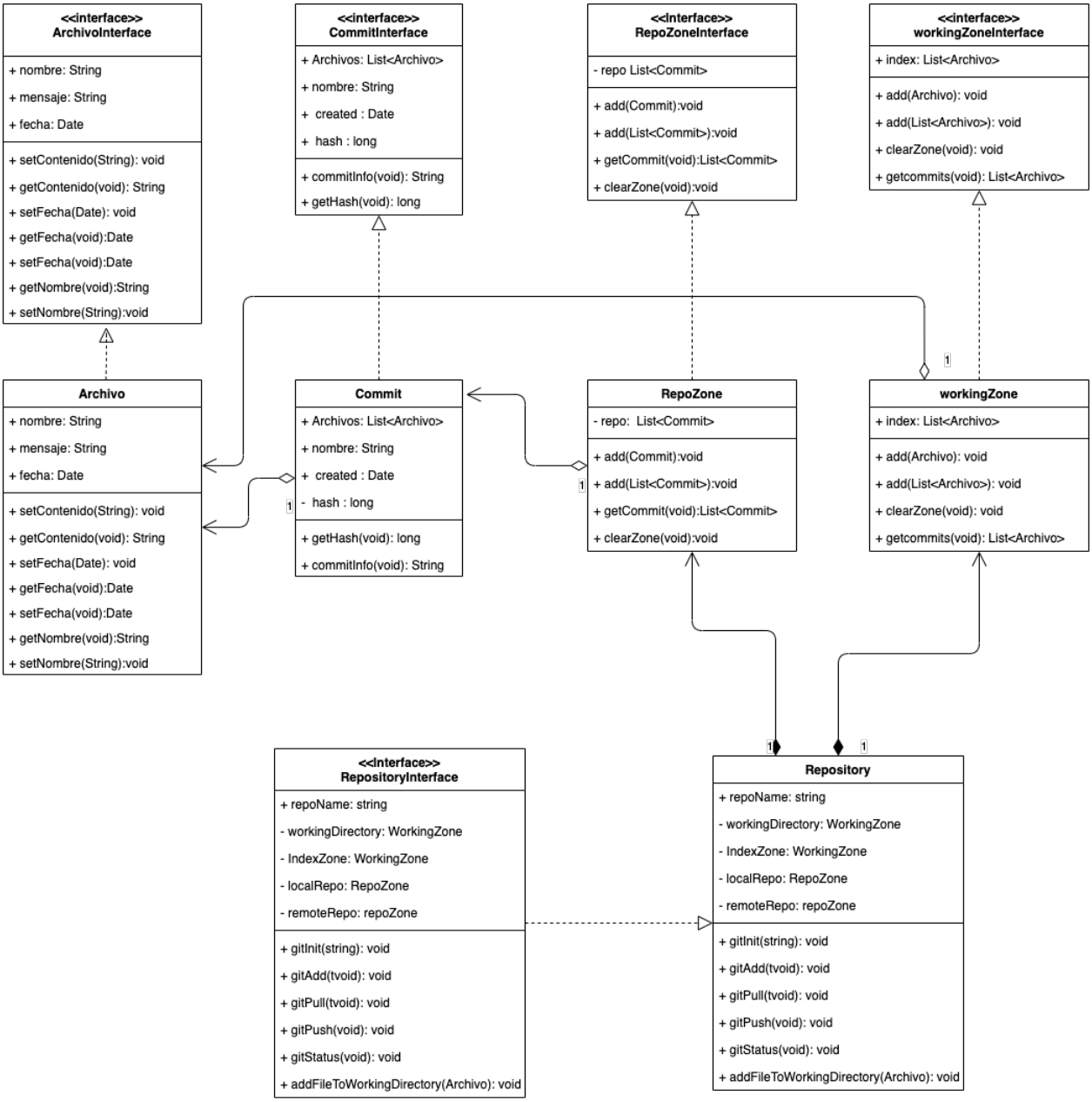
- WorkingDirectory: Espacio de trabajo donde agregaremos los archivos que queramos agregar como cambios en nuestro flujo de trabajo principal.
- IndexZone: Espacio de trabajo donde se preparan los archivos que están listos para ser enviados al nuestra rama de trabajo y que cuentan como cambio oficial dentro de nuestro flujo de trabajo mencionado.
- Local Repository: Espacio de trabajo donde se encuentran nuestros cambios locales, no compartidos con el equipo de trabajo en su totalidad.
- Remote Repositoyry: Espacio de trabajo donde se guardan nuestros cambios globales, estos son compartidos en su totalidad por el equipo de trabajo que utiliza el espacio en cuestión.

A su vez se define una unidad llamada “Commit”, el commit es una entidad que contiene todos los archivos que asociados a un cambios dentro de nuestro trabajo, junto con un mensaje,meta datos y la información correspondiente a la persona que hizo este cambio. Estos commits solo se encuentran en dos espacios de trabajo los cuales son “Local Repository” y “Remote Repository”.

A su vez se desprende la unidad de “Archivo”, el “archivo” es la unidad de minima dentro de un commit y como su nombre lo indica es donde nosotros trabajamos y realizamos cambios en nuestro proyecto en cuestion.

Diseño de solución

Para el diseño en cuestión tenemos el siguiente diagrama UML que muestra la relación entre nuestras clases:



Tenemos 5 clases principales que implementan sus interfaces correspondientes para una mayor comprension entre el contrato que tienen con el diseño, estas son :

- Archivo: Clase que representa un archivo de texto plano dentro de la simulación creada.
- Commit : Clase que representa un cambio dentro de nuestro flujo del trabajo dentro de la simulación. Esta Contiene una Lista de Archivos ya actualizados o agregados en la rama de trabajo.
- WorkingZone: Clase que tiene como objetivo representar las dos primeras estaciones de trabajo de la simulación. "WorkingDirectory" e "Index". Estas áreas solo manejan archivos entre sí y representan los estados de trabajo en progreso y teminado en la simulación.
- RepoZone: Clase que tiene como objetivo representar las siguientes dos estaciones de trabajo de la simulación. "Local Respository" y "RemoteRepository". Estas áreas manejan los commits que se guardan como cambios en el flujo de trabajo de un proyecto.

*Repository : Clase principal que contiene a las otras zonas, permite agregar archivos al "workingDirectory", moverlos al "index", crear commits para el "local Repository" y subir los cambios al "Remote Repository", además de mostrarnos los estados de cada zona en cuestión.

Aspectos de la implementación

Nuestro programa ya creado consta de 6 packages, los cuales son:

- archivo: Contiene la interface de “Archivo” y la definición de clase del mismo.
- commit : Contiene la interace de “Commit” y la defínción de clase del mismo.
- repozone: Contiene la interface “RepoZone” y la definición de clase del mismo.
- workingzone: conitene la interface de “WorkingZone” y la definición de clase del mismo.
- repository: Contiene la interface de “Repository” y la definición de clase del mismo.
- main: Contiene nuestra clase principal, la cual implementa “Repository”, abre un menu interactivo con numeros y nos permite simular la implementación de la solución descripta.

A su vez todo el codigo fuente se concentra en el directorio “src” dentro del directorio “lab3”. Se han ocupado librerías propias del lenguaje para trabajar de mejor manera cada una de las clases ya mencionadas. esta librerías son.

- Date: librería para manejor del tiempo ordenado, utilizado en archivos y commits.
- List: librería para el manejo de listas de cualquier tipo de elemento asociado a estas.
- Instant: libreria utilizada dentro de Commit para crear un hash con respecto al momento en el que este es creado.
- Scanner: utilizado para poder realizar el menú interactivo dentro de la simulación de la implementación y obtener los valores del menu creado.

El proyecto fue creado en un equipo macOS, la que a su vez utilizó como base java version “1.8.0_111”. Otros directorios y archivos como “nbproject” y build.xml forman parte del “ide” utilizado en primera instancia para trabajar en este proyecto pero no son requeridos para poder ejecutarlo.

Instrucciones de uso

Para poder ejecutar el codigo de manera correcta se ha dispuesto de un script de bash llamado "script.sh" dentro del directorio raiz del proyecto. este script crea una carpeta llamada "dist" la que a su vez contiene todos los paquetes ya mencionados , compilados en sus respectivos directorios a "byte code". Una vez terminado esto, el proyecto se desplegará en la terminal en la que se ha ejecutado este script.

Al inicializar este desplegará el menu solicitado para el proyecto, el cual acepta solo numeros enteros. cada numero representa una opcion mostrada ya por pantalla. el no seguir estas instrucciones puede llevar al malfuncionamiento o no ejecución de la simulación del todo.

Las opciones para poder ejecutar el programa son:

- opción n°1: realiza un gitAdd, agrega un elemento al index.
- opción n°2: realiza un gitCommit, crea un commit para pasar los cambios al local repository.
- opción n°3: realiza un gitPull. trayendo los cambios del Remote Repository al Local Repository.
- opción n°4: realiza un gitPush. llevando los cambiios del Local Repository al Remote Repository.
- opción n°5: realiza un gitStatus: mostrando el status de las distintas zonas de trabajo.
- opción n°6: agregar un archivo al working directory.
- opción n°0: terminar la ejecución del programa y salir de este.

Resultados y Autoevaluación

A continuación se listarán los requerimientos completados como resultado de la creación de este simulación.

requerimiento	completado	grado de completacion
gitInit	si	100%
gitAdd	si	100%
gitCommit	si	100%
gitPush	si	100%
gitPull	si	75%
gitStatus	si	100%
addFileToworkingDirectory	si	100%
separacion de espacios de trabajo	si	100%
Archivo class	si	100%
commmit Class	si	100%

Para todos los casos anteriores se probaron creando instancias fuera del menu principal y realizando pruebas pequeñas en la medida de la construcción del software. El único caso no testeado de forma correcta es en gitPush. Cuando "Remote Repository" se encuentra en un estado mucho más avanzado que el LocalRepository, a pesar de que el código sí cuenta con la debida integración para que cubra de manera correcta ese caso.

Conclusiones

Después de terminar con las pruebas correspondientes y ejecución del codigo podemos concluir que la pieza de software emula en su totalidad con lo requerido por el laboratorio en cuestión. Utilizando el paradigma de programcion orientado a objetos como marco de trabajo. Una vez analizado este informe, y tomando las indicaciones realizadas para poder ejecutar el programa. Se puede apreciar que el paradigma utilizado nos permite modelar de manera eficiente cada una de las partes que constituyen el problema explicado.

A su vez el paradigma orientado a objetos nos permite reutilizar codigo, para las distintas zonas de trabajo, haciendo que este se vuelva mucho más eficiente y menos complejo de analisis para usuarios que requieran modificar el codigo fuente del mismo. Por otro lado, el lenguaje utilizado cuenta con herramientas que nos permitieron propias y bibliotecas especificas que nos ayudaron a extender de mejor manera la realización de algunas tareas importantes dentro de la ejecución de este programa.

Referencias

Para la construcción de la pieza de software que hace referencia este informe, se utilizaron las siguientes fuentes bibliograficas:

- Java OOP (Object-Oriented Programming). (s. f.). w3school. https://www.w3schools.com/java/java_oop.asp
- Essentials, Part 1, Lesson 1: Compiling Running a Simple Program. (s. f.). Oracle. https://www.oracle.com/java/technologies/compile.html
- Schildt, Herbert. Java: The Complete Reference, Eleventh Edition. New York: McGraw-Hill Education, 2018.