Emmanuel Isai Chable Colli

FacULTAD DE MATEMATICAS  Arquitectura de software

Documentación framework mvc

# Documentación framework MVC

Contenido

[Documentación framework MVC 0](#_Toc63851632)

[Descripción uso del framework 2](#_Toc63851633)

[Descripción de componentes 8](#_Toc63851634)

[Descripción de clases 12](#_Toc63851635)

[Descripción de las secuencias 24](#_Toc63851636)

[Video explicativo del ejemplo: 26](#_Toc63851637)

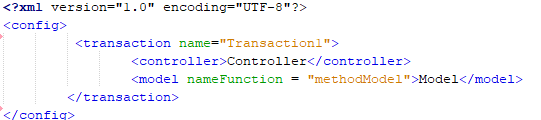
[Autor: **¡Error! Marcador no definido.**](#_Toc63851638)

## Descripción uso del framework

Objetivo principal:

Representar el patrón arquitectónico MVC, sin notarse explícitamente en el código, el framework consiste en ejecutar un método de una clase que usted considere parte del modelo o el modelo mismo conociendo el nombre completo de la clase y el método en específico a ejecutar, cabe destacar que en dicho método del modelo podrá acceder al controlador que se tenga definido en el archivo.

Toda esta información estará incluida en un archivo de configuración de tipo XML como el siguiente:



Funcionalidades:

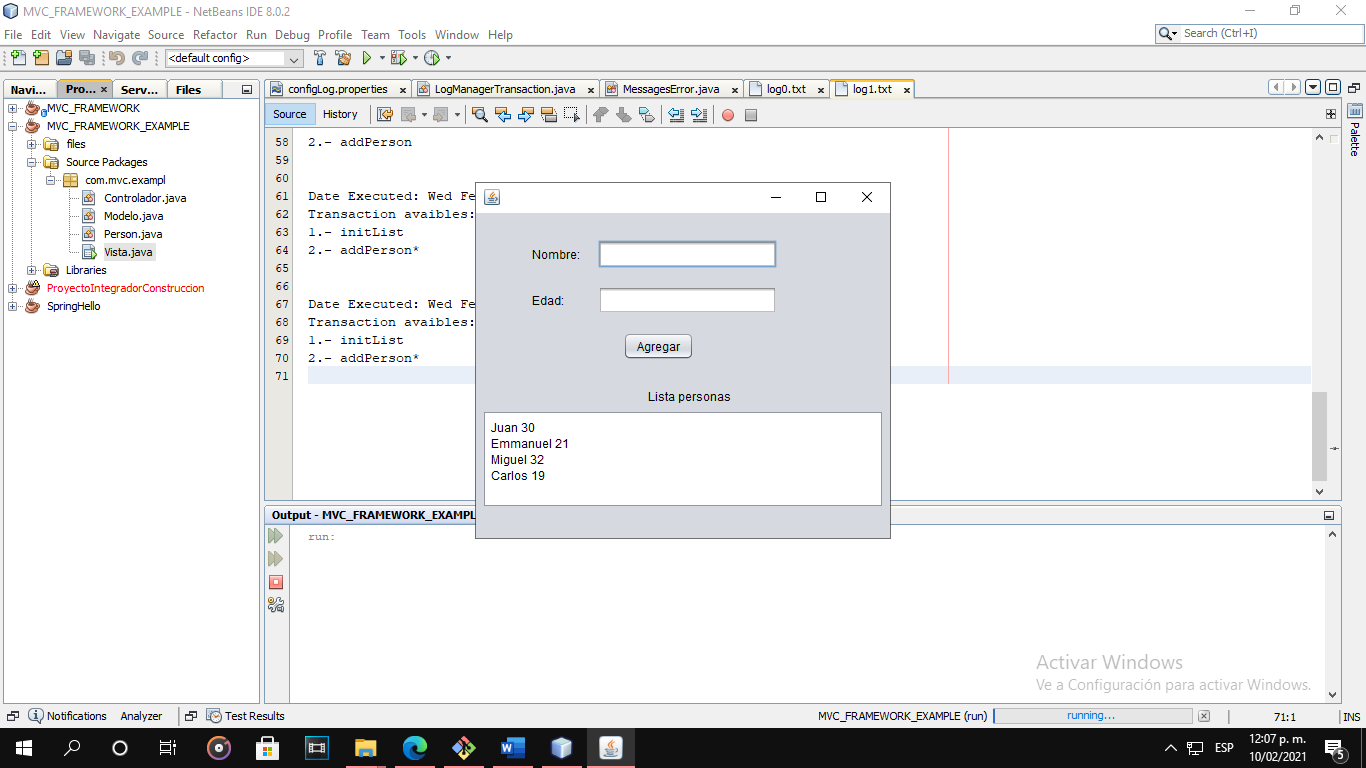
1. Ejecutar una transacción definida en el archivo de configuración por su nombre definido en el atributo name
2. Ejecutar método de una clase modelo la cual dentro de las etiquetas <model>nombre de la clase</modelr> se encontrará el nombre completo de la clase la cual hace referencia al modelo y el atributo nameFunction sirve para especificar que método se ejecutará en base a su nombre.
3. Utilizar la clase controlador-definida en <controller>nombre de la clase</controller> dentro del método del modelo.
4. Definir el comportamiento de un archivo bitácora, mediante el archivo *configLog.properties*
5. Guardar información de la ejecución de las transacciones.

Ejemplo de uso:

Clase vista



Una clase vista, generada con el drop and drag de NetBeans IDE, la ejecución es la siguiente:



Clase modelo



Clase Persona



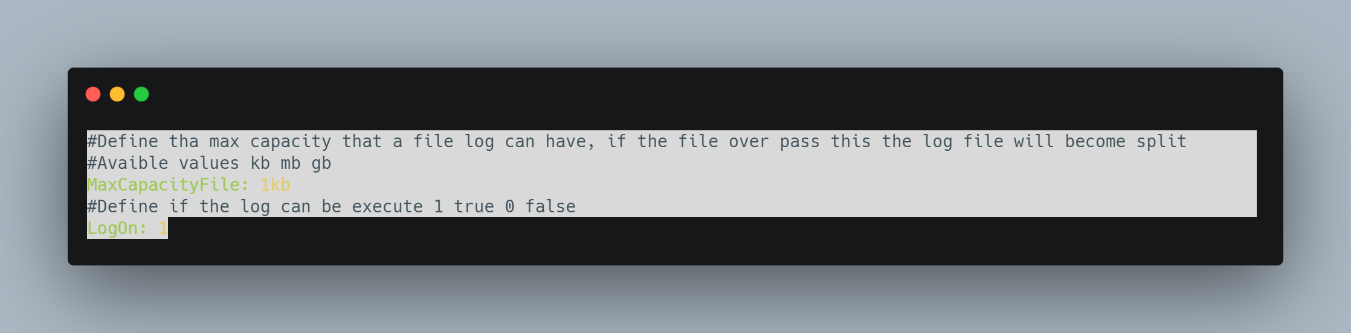
Clase controlador



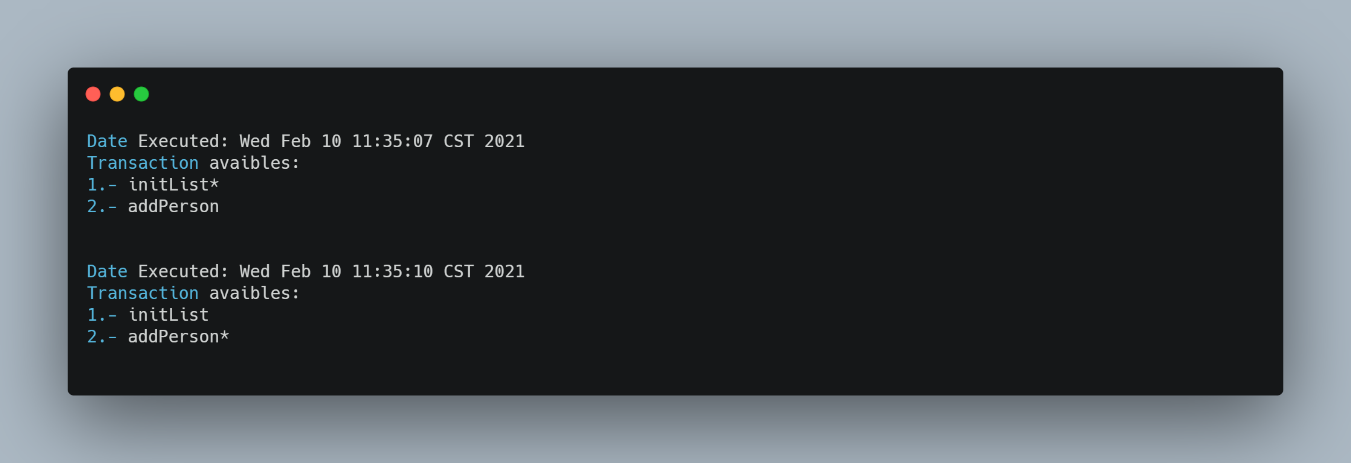
Archivo de configuración



Archivo de configuración del log



Contenido del archivo log



## Descripción de componentes

*Nombre:* ManagerReflection

*Descripción:* Componente encargado de ejecutar los métodos de las clases controlador y modelo mediante la API de Java “Reflection”

Dependencias con otros componentes: Ninguno

*Interfaces de Salida:*

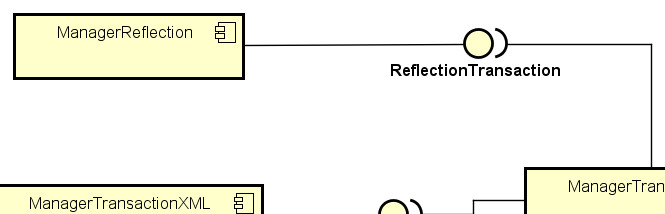
Interface ReflectionProcessor

* getInstanceClass
* runMethodModel

Interfaces de Entrada: Ninguna

Artefactos: API Reflection java

*Diagrama:*



*Nombre:* ManagerXMLTransaction

*Descripción:* Componente encargado de leer la información del archivo de configuracion config.xml, transforma todas las transacciones definidas a una lista de objetos de tipo Transaction

Dependencias con otros componentes: Ninguno

*Interfaces de Salida:*

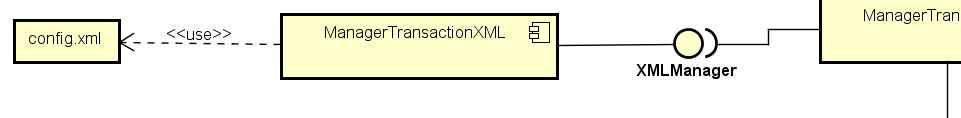
Interface XMLManager

* readXMLFile

Interfaces de Entrada: Ninguna

Artefactos: archivo config.xml

*Diagrama:*



*Nombre:* LogManagerTransaction

*Descripción:* Componente encargado de administrar el archivo log#.txt el cual escribe en un formato sencillo la fecha de ejecución de una transacción, la transacción ejecutada y todas las transacciones disponibles.

Por defecto el nombre del primer archivo es log0.txt

De igual manera revisa de un archivo de configuración el tamaño máximo del archivo y si se encuentra activado el log, si un archivo sobrepasa el límite de tamaño se divide en otro archivo el cual cambia a ser llamada log1.txt cada vez que se sobrepasa avanza en una unidad su nombre log2.txt…logn.txt.

Dependencias con otros componentes: Ninguno

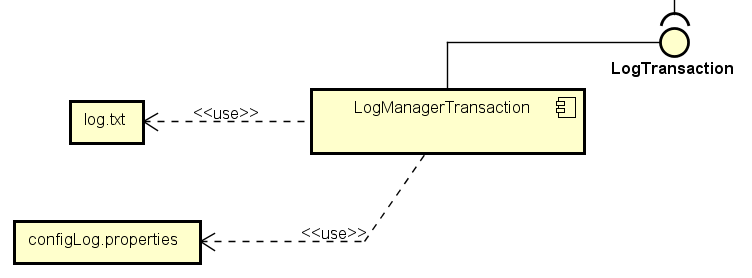
*Interfaces de Salida:*

Interface LogTransaction

* writeLogFile

Interfaces de Entrada: Ninguna

Artefactos: archivos log#.txt, configLog.properties

*Diagrama: *

*Nombre:* ManagerTransactions

*Descripción:* Componente encargado de ejecutar una transacción que se solicite por su nombre, el flujo es muy sencillo:

1. Encuentra la transacción solicitada.
2. Genera una instancia del controlador definido.
3. Luego invoca el método de una clase considerada modelo, que pasa como argumento, la vista donde se ejecuta la transacción, el controlador y algún objeto como argumento.
4. Para después guardar en el log la información de la transacción realizada.

Dependencias con otros componentes: ManagerReflection, ManagerXMLTransaction, ManagerLog

*Interfaces de Salida:*

Interface TransactionExecutor

* executeTransaction

Interfaces de Entrada:

Interface ReflectionTransaction

* getInstanceClass
* runMethodModel

Interface XMLManager

* readXMLFile

Interface LogTransaction

* writeLogFile

Artefactos: ninguno

*Diagrama:*

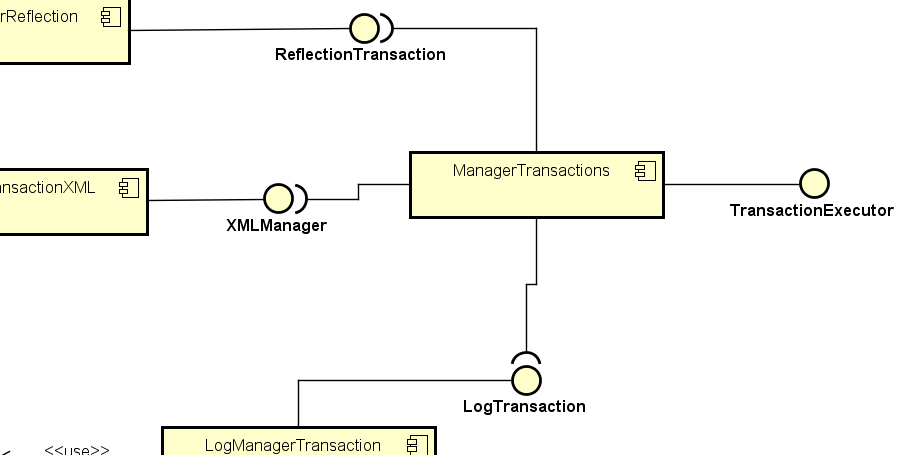
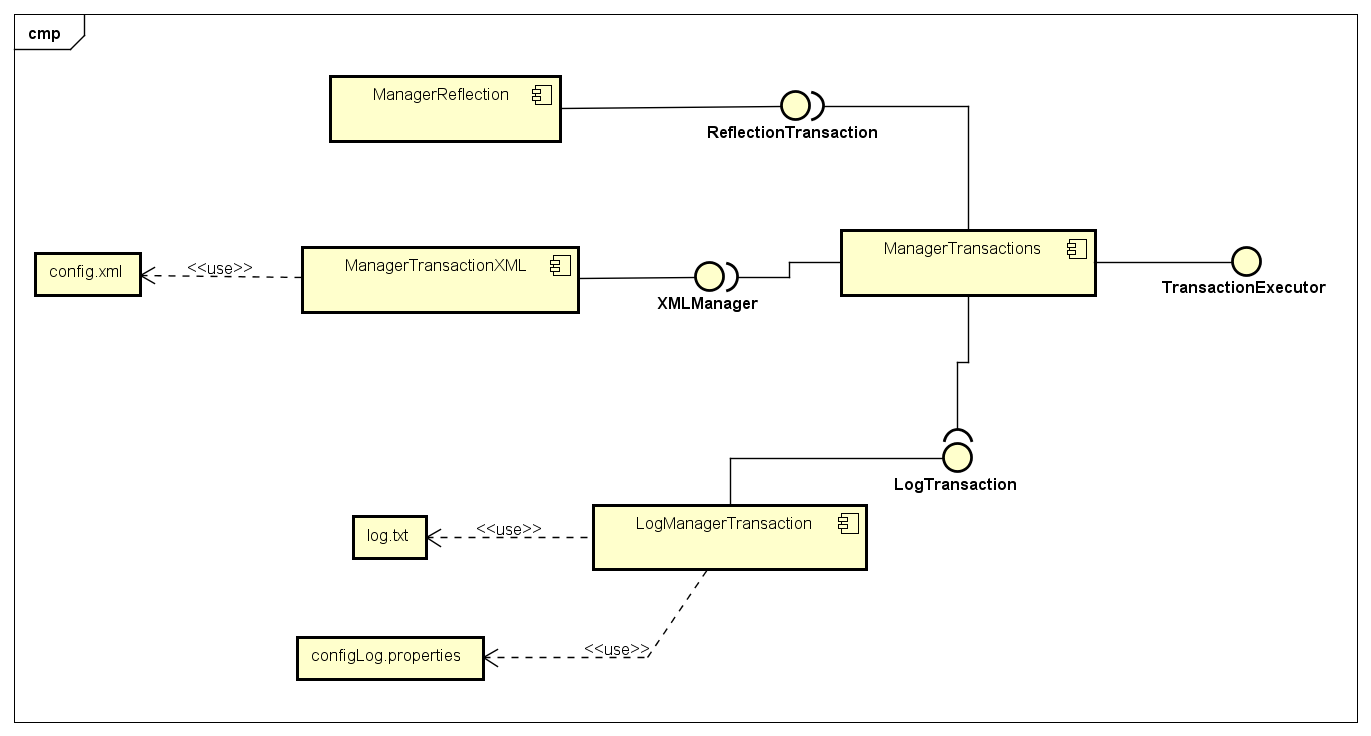
**

Diagrama total de componentes:

**

## Descripción de clases

Paquete reflection

*Nombre de la clase:* ManagerReflection

*Descripción:* Clase encargada de crear una instancia de una clase con su nombre, e igual ejecutar un método de una clase modelo.

*Dependencias con otras clases:*

* Implementación ReflectionTransaction

*Atributos:*

1. Instances

Tipo: HashMap<String, Object>

Valor: HashMap instanciado, pero vacío.

Visibilidad: Private

Descripción: Atributo utilizado para guardar instancias de las clases creadas utilizando esta clase.

*Funciones:*

1. ManagerReflection

Tipo: void

visibilidad: public

descripción: Constructor de la clase

1. getInstance

Tipo: Object

Visibilidad: public

Argumentos:

* nameClass: String

Descripción: Método con la implementación de la interfaz ReflectionTransaction para generar una instancia de una clase con su nombre

1. runMethodModel

Tipo: void

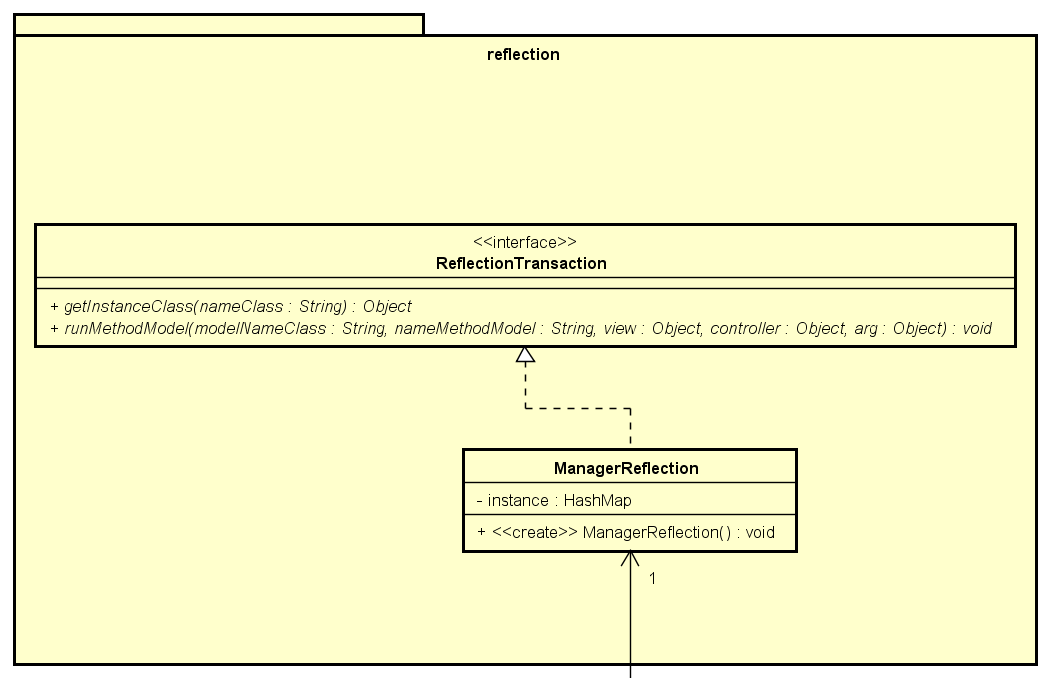
visibilidad: public

Argumentos:

* modelNameClass : String
* nameMethodModel : String
* view : Object
* controller : Object
* arg : Object

Descripción: Método con la implementación de la interfaz ReflectionTransaction para crear una instancia de una clase modelo, y ejecutar el método de esa clase, con los parámetros que sean necesarios.

Diagrama:



Paquete logger

*Nombre de la clase:* ManagerFiles

*Descripción:* Clase encargada de ejecutar funciones relacionadas con archivos, escritura, lectura y el conocer los tamaños de los archivos con diferentes magnitudes kb, mb, gb

*Dependencias con otras clases:*

* none

*Atributos:*

1. *none*

*Funciones:*

1. ManagerFiles

visibilidad: public

Constructor de la clase

1. readFile

Tipo: String

Visibilidad: public

Argumentos:

* path: String

Descripción: Método encargado leer el contenido de un archivo, devolviendo un dato de tipo String con todo el contenido.

1. writeFile

Tipo: void

visibilidad: public

Argumentos:

* path: String
* content : String

Descripción: Método encargado de escribir un contenido dentro de un arhcivo.

1. getFileSizeGigaBytes

Tipo: double

Visibilidad: public

Argumentos:

* file: File

Descripcion: Método encargado de obtener el tamaño de un archivo en gigabytes

1. getFileSizeMegaBytes

Tipo: double

Visibilidad: public

Argumentos:

* file: File

Descripcion: Método encargado de obtener el tamaño de un archivo en megabytes

1. getFileSizeKiloBytes

Tipo: double

Visibilidad: public

Argumentos:

* file: File

Descripcion: Método encargado de obtener el tamaño de un archivo en kilobytes

*Nombre de la clase:* LogManagerTransaction

*Descripción:* Clase encargada de administrar los archivos de la bitocora, configuración y la bitácora misma, se encarga de definir el formato para la escritura de un archivo que será bitácora del framework

*Dependencias con otras clases:*

* Implementación LogTransaction
* Agregación ManagerFiles

*Atributos:*

1. pathLog

Tipo: String

Visibilidad: private

descripción: Atributo utilizado para crear, leer o escribir un archivo de bitácora ya que este representa la ruta del archivo.

1. PROPERTY\_LOG\_ON

Tipo: String

Visibilidad: private

Valor: “LonOn”

Descripción: representa el nombre de la propiedad correcta, para activar la función de generar bitácoras

1. PROPERTY\_MAX\_CAPACITY

Tipo: String

Visibilidad: private

Valor: “MaxCapacity”

Descripción: representa el nombre de la propiedad correcta, para limitar el tamaño de archivos

1. VALUES\_IS\_LOG\_ON

Tipo: String

Visibilidad: private

Valor: “1”

Descripción: representa el valor de la propiedad correcta, para activar la función de generar bitácoras.

1. props

Tipo: Properties

Visibilidad: private

Descripción: representa las propiedades de la configuración para la bitácora

1. managerFiles

Tipo: ManagerFiles

Visibilidad: private

Descripcion: representa una instancia de la clase ManagerFiles, la cual sirve para ejecutar varias características de un log.

1. currentNumberFile

Tipo: int

Visibilidad: private

Descripción: representa el número actual para el log

1. currentFile

Tipo: File

Visiblidad: private

Descripcion: representa el archivo actual del log, con un objeto File de Java

*Funciones:*

1. LogManagerTransaction

Tipo: void

Visibilidad: public

Descripción: Constructor de la clase, inicializa varios de los atributos de la clase

1. isLogOn

Tipo: boolean

Visibilidad: private

Argumentos:

* none

Descripción: verifica si el valor de la propiedad es el valor que representa la activación del log

1. checkFileSize

Tipo: void

visibilidad: private

Argumentos:

* none

Descripción: Encargado de revisar el tamaño actual del archivo log, si el archivo sobrepasa el límite puesto en el archivo .properties entonces se generará otro archivo log.

1. changeCurrentNumberFile

Tipo: void

Visibilidad: private

Argumentos:

* none

Descripción: Encargado de cambiar el nombre del archivo en base al número actual de archivo.

1. checkCurrentFile

Tipo: void

Visibilidad: private

Argumentos:

* none

Descripción: Encargado de revisar si el número de archivo del log está creado, de no ser así genera dicho archivo.

1. readFile

Tipo: String

Visibilidad: private

Argumentos:

* none

Descripción: Encargado de leer el contenido actual de un log

1. configLog

Tipo: void

Visibilidad: private

Argumentos:

* none

Descripción: Encargado de leer el contenido del archivo .properties y guardarlo en el objeto props de la clase.

1. writeLogFile

Tipo: void

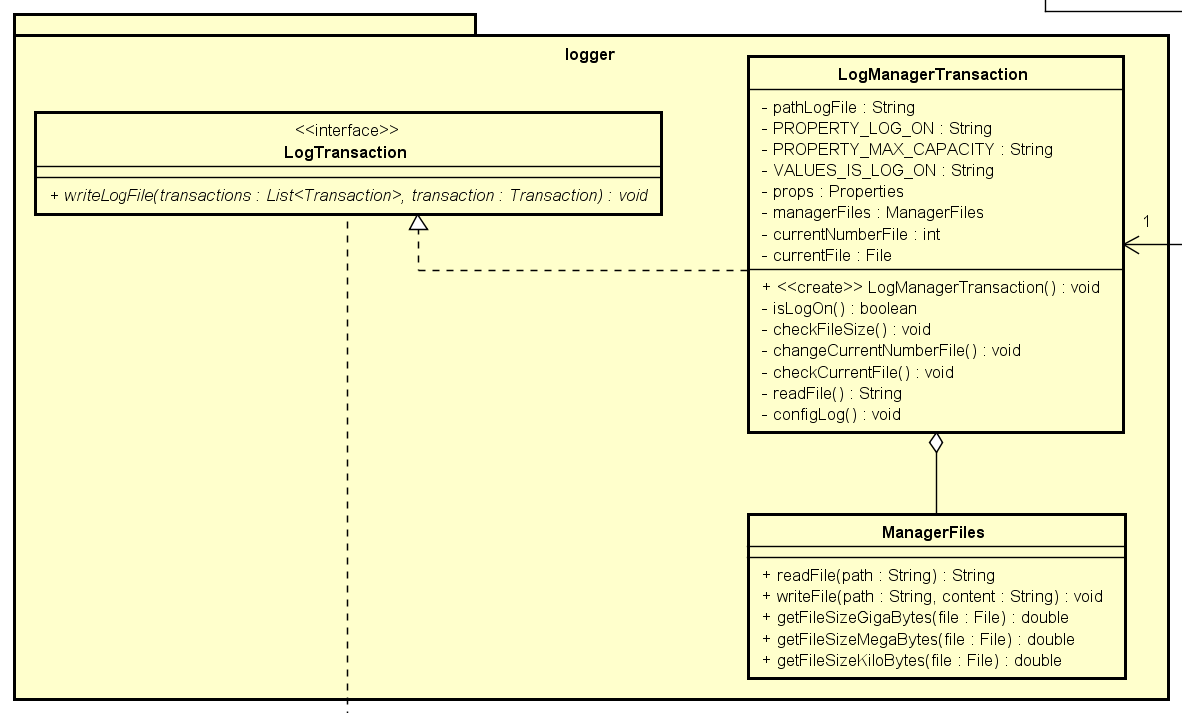
Visibilidad: public

Argumentos:

* transactions:List<Transaction>
* transaction: Transaction

Descripción: implementación del método del mismo nombre con la interfaz LogTransaction

Diagrama:



Paquete xml

*Nombre de la clase:* ManagerTransactionXML

*Descripción:* Clase encargada de leer el archivo de configuración config.xml el cual transforma ese archivo plano en una lista de objetos de tipo transaction.

*Dependencias con otras clases:*

* Implementación XMLManager

*Atributos:*

1. configFile

Tipo: String

Visibilidad: Private

descripción: utilizado para representar la ruta del archivo de configuración, este se inicializa en el constructor

1. NAME

Tipo:String

Valor: “name”

Visibilidad: private

Descripcion: utilizado para identificar dentro de la estructura xml el nombre de una transacción

1. TRANSACTION

Tipo:String

Valor: “transaction”

Visibilidad: private

Descripcion: utilizado para identificar dentro de la estructura xml las etiquetas transaction

1. CONTROLLER

Tipo:String

Valor: “controller”

Visibilidad: private

Descripción: utilizado para identificar dentro de la estructura xml las etiquetas controller

1. MODEL

Tipo:String

Valor: “model”

Visibilidad: private

Descripción: utilizado para identificar dentro de la estructura xml las etiquetas model

1. MODEL\_NAME\_FUNC

Tipo:String

Valor: “nameFunction”

Visibilidad: private

Descripción: utilizado para identificar dentro de la estructura xml el nombre del método del modelo a ejecutar.

*Funciones:*

1. ManagerReflection

Tipo: void

visibilidad: public

Argumentos:

* configFile: String

Descripción: Constructor de la clase inicializa ciertos parametros

1. isCorrectTransaction

Tipo: boolean

Visibilidad: private

Argumentos:

* transaction: Transaction

Descripción: encargado de revisar si un objeto transaction fue creado sin ningún problema.

1. readXMLFile

Tipo: List<Transaction>

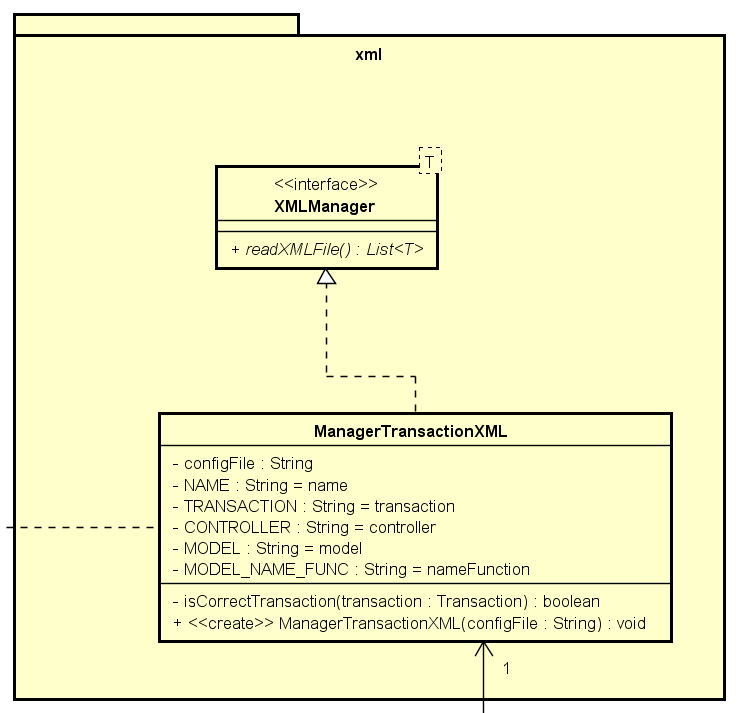
visibilidad: public

Argumentos:

* none

Descripción: Implementación del método readXMLFile de la interfaz XMLManager

Diagrama:



Paquete transaction

*Nombre de la clase:* Transaction

*Descripción:* Abstracción de una trasaction a una clase

*Dependencias con otras clases:*

* none

*Atributos:*

1. name

Tipo: String

Visibilidad: Private

Descripción: Utilizado para representar el nombre de una transacción

1. controller

Tipo: String

Visibilidad: Private

Descripción: Utilizado para representar el nombre de la clase controlador de una transacción

1. model

Tipo: String

Visibilidad: Private

Descripción: Utilizado para representar el nombre de la clase modelo de una transacción

1. modelFunction

Tipo: String

Visibilidad: Private

Descripción: Utilizado para representar el nombre de la función de una clase modelo de una transacción

*Funciones:*

1. Transaction

Tipo: void

Visibilidad: public

Descripción: Constructor de la clase

1. Transaction

Tipo: void

Visibilidad: public

Argumentos:

* name: String
* model: String
* controller: String
* modelFunction: String

Descripción: Constructor de la clase, inicializando todos los atributos.

*Nombre de la clase:* ManagerTransaction

*Descripción:* Clase encargada de ejecutar una transacción definida en el archivo de configuración.

*Dependencias con otras clases:*

* Implementación TransactionExecutor
* Dependencia Transaction
* Asociación 0 a 1 con ManagerReflection
* Asociación 0 a 1 con ManagerXMLTransaction
* Asociación 0 a 1 con LogManagerTransaction

*Atributos:*

1. managerRe

Tipo: ReflectionTransaction

Visibilidad: Private

Descripción: Instancia de la clase ManagerReflection limitando a utilizar los servicios proveído por la interfaz RefectionTransaction.

1. managerXML

Tipo: XMLManager

Visibilidad: Private

Descripción: Instancia de la clase ManagerXMLTransaction limitando a utilizar los servicios proveído por la interfaz XMLManager

1. managerLog

Tipo: LogTransaction

Visibilidad: Private

Descripción: Instancia de la clase LosMangerTransaction limitando a utilizar los servicios proveído por la interfaz LogTransaction

1. PATH\_CONFIG\_FILE

Tipo: String

Visibilidad: Private

Descripción: Constante con el valor de la ruta específica del archivo de configuracion

1. ERROR\_NO\_TRANSACTION\_EXISTS

Tipo: String

Visibilidad: Private

Descripción: Mensaje de error por si la transacción no existe a la hora de llamar a esa transacción.

*Funciones:*

1. ManagerTransaction

Tipo: void

visibilidad: public

descripción: Constructor de la clase inicializa los parámetros de la clase.

1. executeTransaction

Tipo: void

Visibilidad: public

Argumentos:

* nameTransaction: String
* frame: JFrame
* arg: Object

Descripción: Método con la implementación de la interfaz TransactionExecutor, para el método executeTransaction

Diagrama:

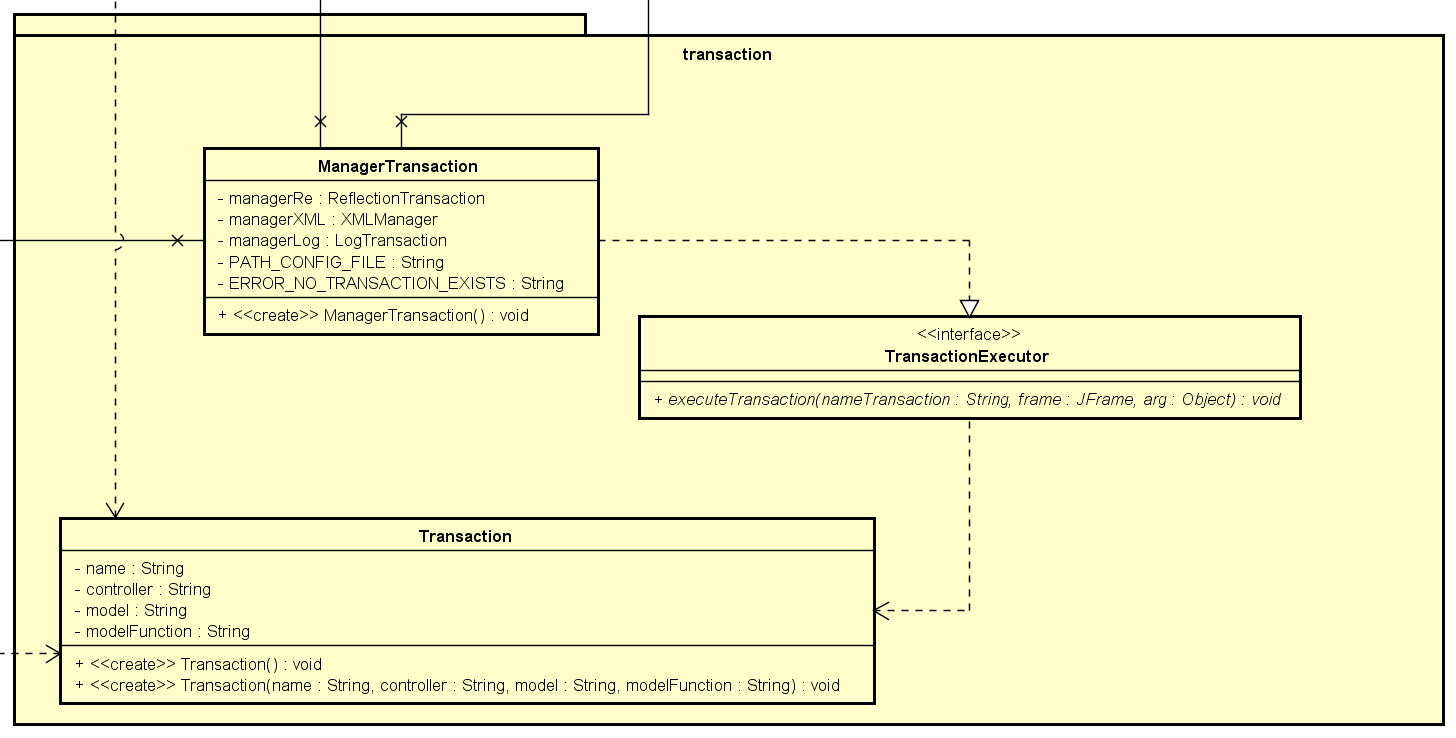
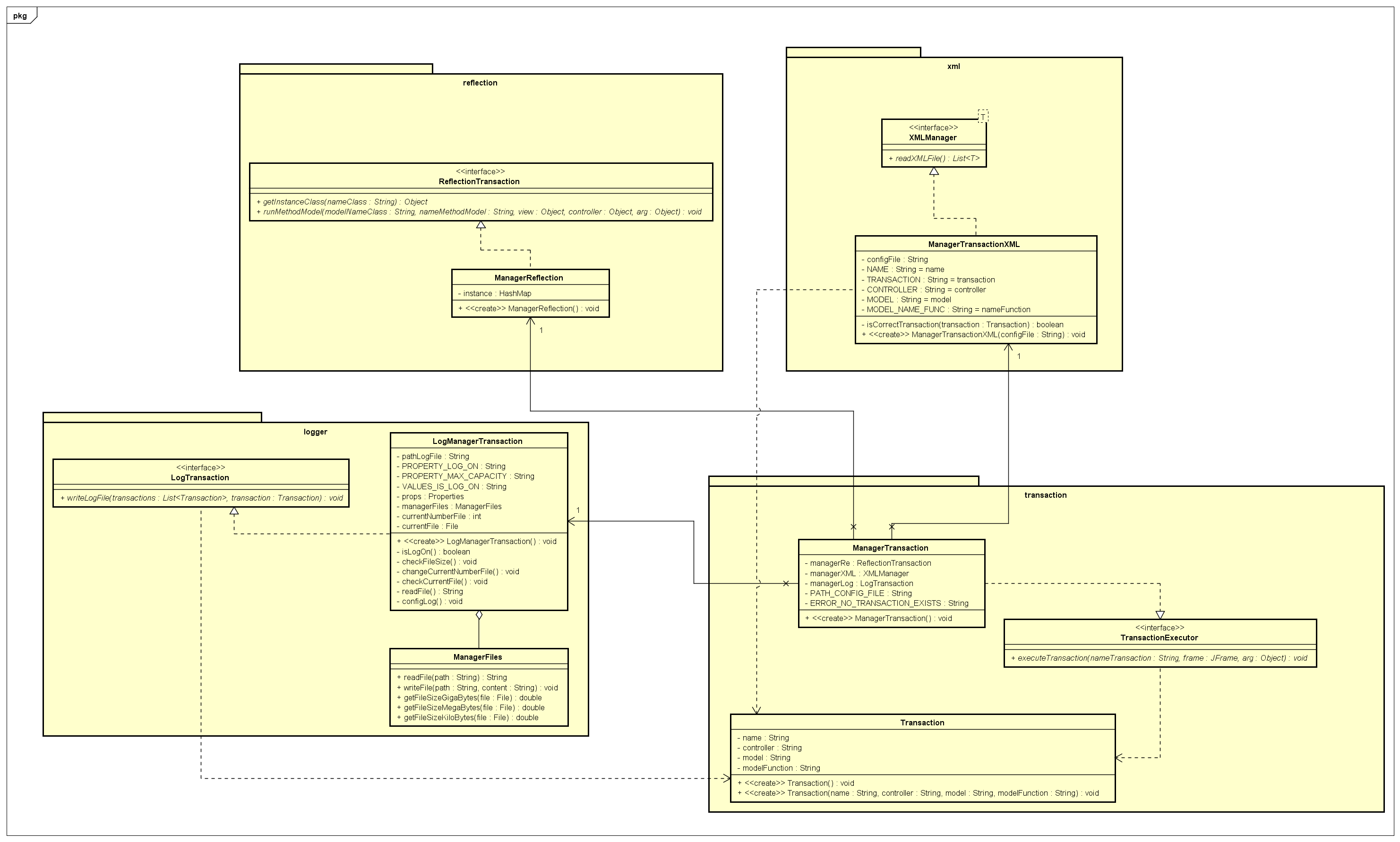


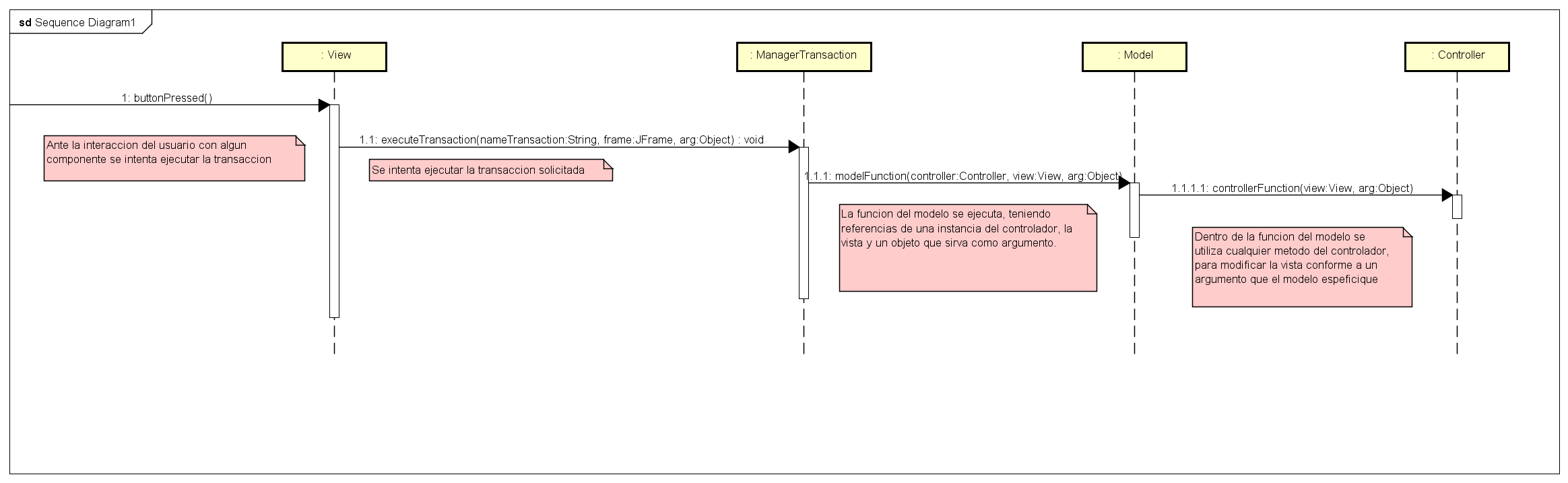
Diagrama de clases



Cabe destacar que se utilizaron excepciones guardadas en su respectivo paquete, igual se utilizaron enums para guardar ciertas constantes, como son mensaje de errores, identificadores de las propiedades, etc.

## Descripción de las secuencias

Diagrama se secuencia orientado a una implementación de un usuario

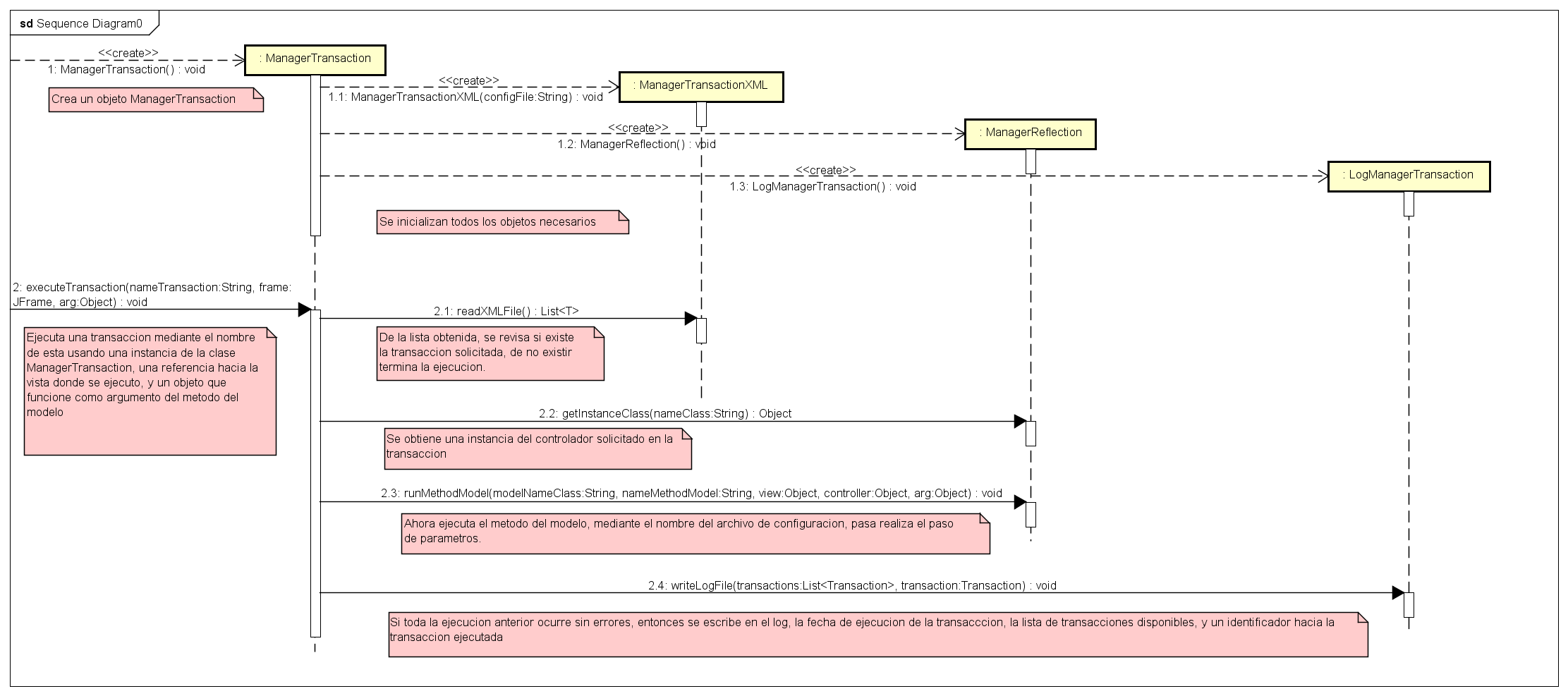


La intención es que de una clase vista generada por el usuario del framework pueda administrar las interacciones de usuarios (ActionPerformed, etc.) en donde tenga una instancia de ManagerTransaction, de esta manera podrá ejecutar una transacción desde la vista.

Se debe tener ya las clases modelo y controlador que se utilizarán, estas dos deben usar el constructor por defecto (sin parámetros), el controlador podrá definir las funciones que se deseen utilizar (actualizar modelo, actualizar componentes de vista, visualizar una nueva vista, etc).

El modelo deberá tener un constructor por defecto y métodos más específicos los cuales tengan como parámetros al menos un controlador (la clase creada por el usuario) y vista (esta puede ser la misma instancia por donde se ejecuta una transacción) para aquellos métodos que se ejecutarán con una transacción.

Diagrama de secuencia para el flujo de actividades de todas las clases del framework



La secuencia entre los componentes del framework es un poco más larga, pero es bastante sencilla, de una instancia de ManagerTransaction, se ejecuta el método executeTransaction dentro de este se hace uso de las instancias de ManagerTransactionXML, ManagerReflection y LogManagerTransaction, para primero leer del archivo de configuración todas las transacciones definidas, luego localizar la transacción solicitada, después generar una instancia del controlador, para luego generar una instancia del modelo y ejecutar el método correspondiente y finalmente registrar está ejecución en los archivos log.txt

## Video explicativo del ejemplo:

Video de ejecución del ejemlo: <https://youtu.be/diAsqQXM4Gg>

Video explicativo del framework: <https://youtu.be/CFGNGKDc5X8>

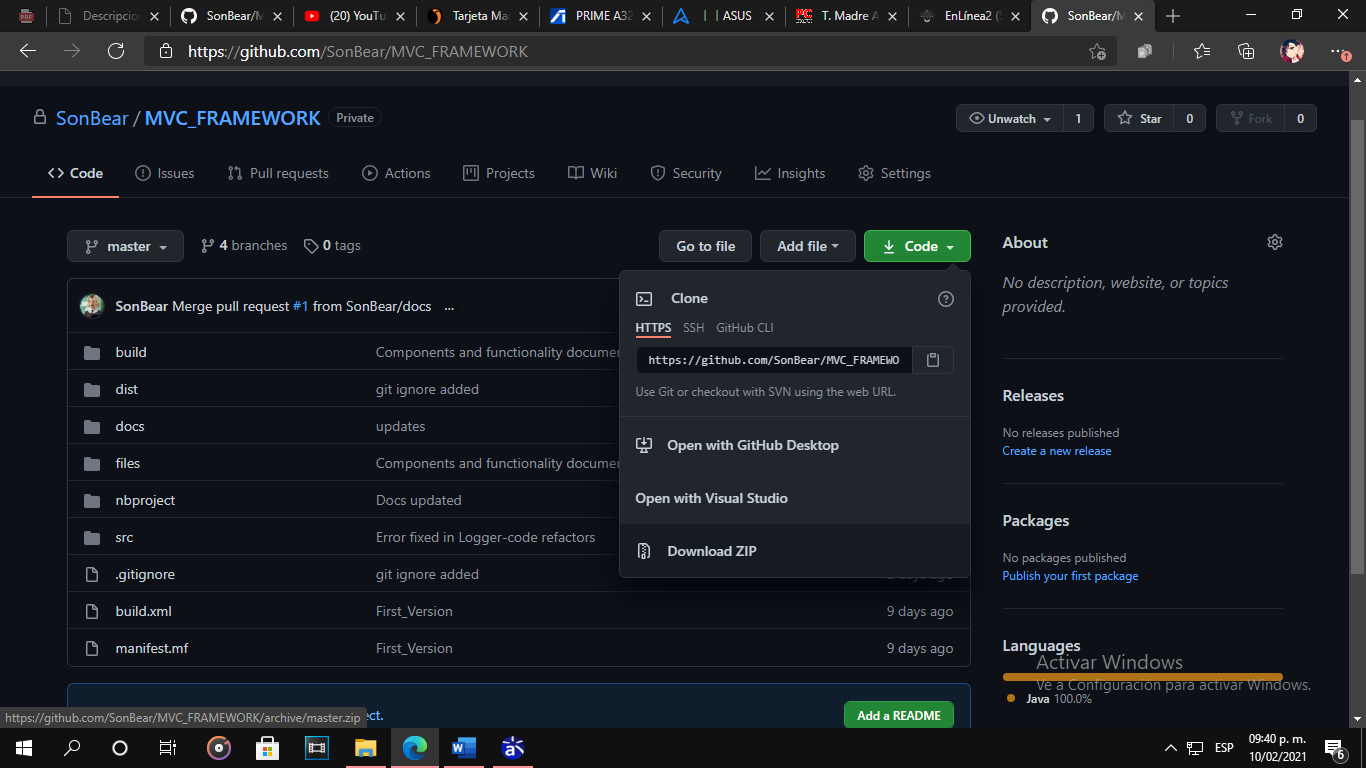
Se dividió de esta manera los videos debido a un error al grabar el video explicativo

## Repositorio Git-Hub

Para descargar los archivos del proyecto puede consultar la siguiente liga al repositorio de git-hub correspondiente:

<https://github.com/SonBear/MVC_FRAMEWORK.git>

Ya estando en la pagina del repositorio es posible descargar todos los archivos directamente en un archivo comprimido:



En otro caso es posible clonar el repositorio mediante git-hub desktop.

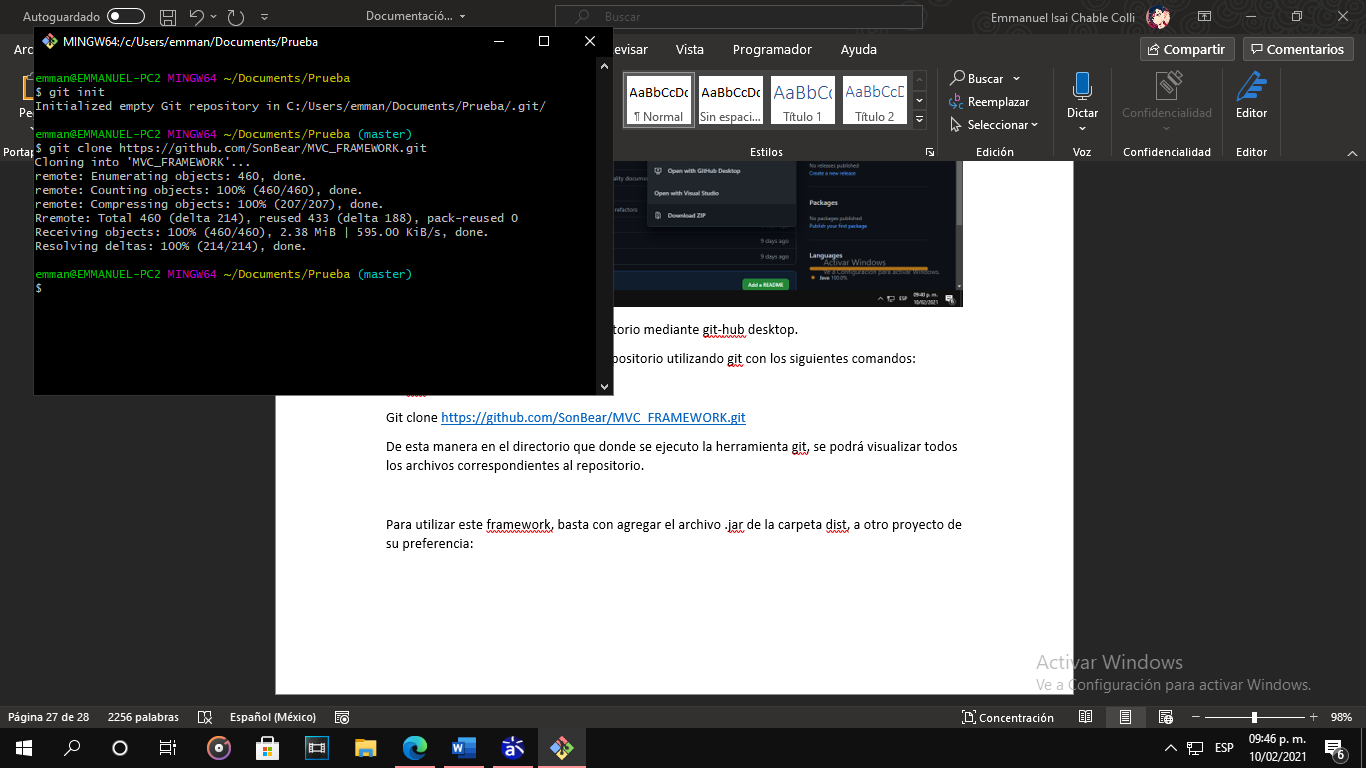
De igual manera es posible clonar el repositorio utilizando git y la consola git-bash con los siguientes comandos:

Git init

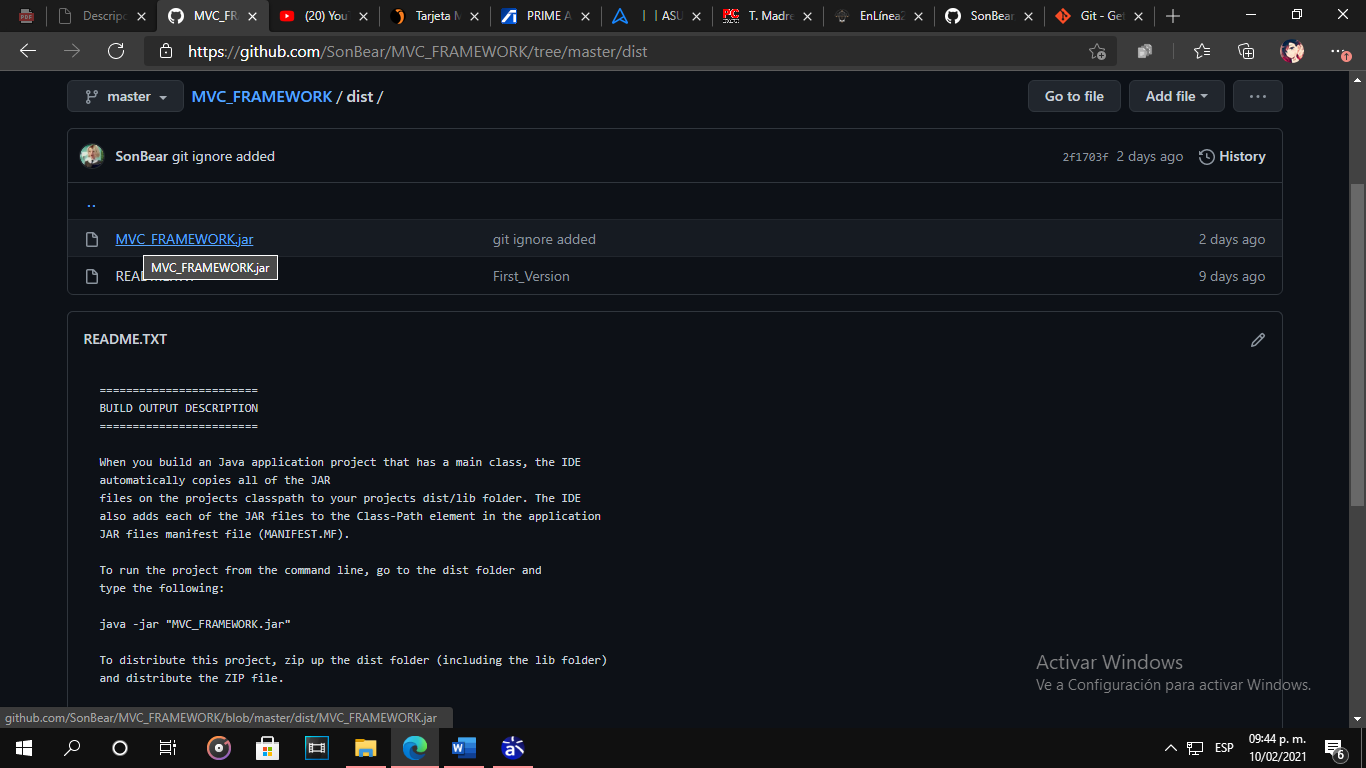
Git clone <https://github.com/SonBear/MVC_FRAMEWORK.git>

Dirección HTTPS mostrada en la imagen.

De esta manera en el directorio que donde se ejecutó la consola git-bash, se podrá visualizar todos los archivos correspondientes al repositorio.



Para utilizar este framework, basta con agregar el archivo .jar de la carpeta dist, a otro proyecto de su preferencia:



Archivos del ejemplo:

Ingrese a la rama ejemplo de uso, en github y descargue los archivos de cualquiera de las formas mencionadas anteriormente.