75.10 – Técnicas de Diseño

TP 1

1º Cuat. 2014

Grupo 8

90697 - Eisner, Ariel

89563 - Ferreyra, Oscar

Correcciones

- **Los comentarios se mantuvieron en castellano debido al poco manejo que tenemos sobre el idioma ingles y al ser un comentario y no una exigencia**
- **Se quitó el prefijo "m_" a todos los atributos de clase**
- **Se usa foreach en vez de for**
- **Se arreglaron incongruencias respecto al uso de la clase LevelLog y se implementó una clase Level que es la que maneja la autorización del logueo de los mensajes**
- **Printer pasa a ser una clase abstracta**
- **Se cambió interface métodos para loguear, se implementa como se sugirió en la correcion**
- ¿Cuál es la ventaja de separar los patterns en diferentes clases: PatternSeparator, PatternMethodName, etc, que motivo a modelarlo asi? Que ventajas, desventajas?
  Lo que nos motivó a modelarlo de esa manera fue el hecho de que si en el futuro algún pattern modifica su comportamiento, el cambio a realizar sobre el código estaría encapsulado solamente en esa clase sin necesidad de tocar otras clases.
- **La manera de cerrar el loguer se mantiene, suponemos que el usuario de la API va a leer la documentación y con remarcar esta particularidad se estaría solucionando la situación planteada**
- **La configuracion del Loguer se hace directamente en el archivo de properties, la clase Configurator lo único que hace es levantarla y pasarsela a la Clase Loguer**

A CONTINUACION SE ACTUALIZA LA DOCUMENTACION DE LA API EN FUNCION DE LOS CAMBIOS PREVIAMENTE DETALLADOS

La funcionalidad de todas las clases están comentadas en sus respectivos códigos

Uso de la API para configurar la aplicación

La configuración de las consolas o archivos sobre los cuales se loguearan los mensajes se realiza directamente sobre un archivo cuyo path debe ser pasado por parámetro al instanciar la clase Configurator, la misma se encargar de parsear dicho archivo y luego pasarle la configuración a la clase Logger, todo esto será detallado a continuación

Configurando el archivo Properties

La estructura del archivo properties debe ser CLAVE = VALOR y hay dos claves que deben estar, las claves “files” y la clave “consoles”
Cada una albergara el nombre de las salidas a través de las cuales loguearan sus mensajes usando como separador de campo la “,”

Ejemplo

files=file4.txt,file2.txt,
consoles=SUPER_CONSOLA,

Configurando propiedades de consolas y archivos

Hay cuatros propiedades que deber ser configuradas

El separador de mensaje, atributo “separator”
El formato de hora , atributo “formatDate”
El nivel de mensaje (DEBUG,INFO,ETC), atributo “logLevel”
El formato en si del mensaje, atributo “format”

Para configurar un atributo de una consola o archivo en particular hay que usar como clave el nombre del archivo o consola, seguido de un "-" y luego el nombre del atributo a modificar, luego el signo "=" y luego el valor a setear. Ejemplo

files=file1.txt,
consoles=SUPER_CONSOLA,
SUPER_CONSOLA-**logLevel**=FATAL
SUPER_CONSOLA-**formatDate**= HH\:mm\:ss
SUPER_CONSOLA-**format**=%d-%n-%m-%n-%p-%n-%t-%n-%F-%M-%t
SUPER_CONSOLA-**separator**=-
File1.txt-**format**=%d-%n-%m-%n-%p-%n-%t-%n-%F
File1.txt-**separator**=-
File1.txt-**formatDate**= yyyy/MM/dd HH:mm:ss
File1.txt-**format**=%d-%n-%d-%n-%m-%n-%p-%n-%t-%n-%F-%M

Integracion de la configuración a la clase Logger

Una vez finalizada la configuración de las distintas impresoras a utilizar, hay que informarle a la clase Logger cuáles son estas, para ello se instancia la clase Logger pasándole como parámetro la estructura de datos devuelta por el método getPrintersConfiguration() (DICHA ESTRUCTURA SE DETALLA EN LA SECCION DE DISEÑO) y luego de ello la herramienta ya queda lista para poder ser utilizada. Ejemplo

```
Configurator configurator = new Configurator(PATH-FILE-PROPERTIES);
Logger loguer = new Logger( configurator.getPrintersConfiguration() )
loguer.logError("MensajeError");
```

Uso de la API de Logger

Hay 6 niveles *de log los cuales son: OFF*, *FATAL*, *ERROR*, *WARN*, *INFO*, *DEBUG*

Para realizar cualquier mensaje de log, hay que intanciar la clase Logger y luego llamar el método logSUFIJO_NIVEL_LOG(Mensaje), por ejemplo

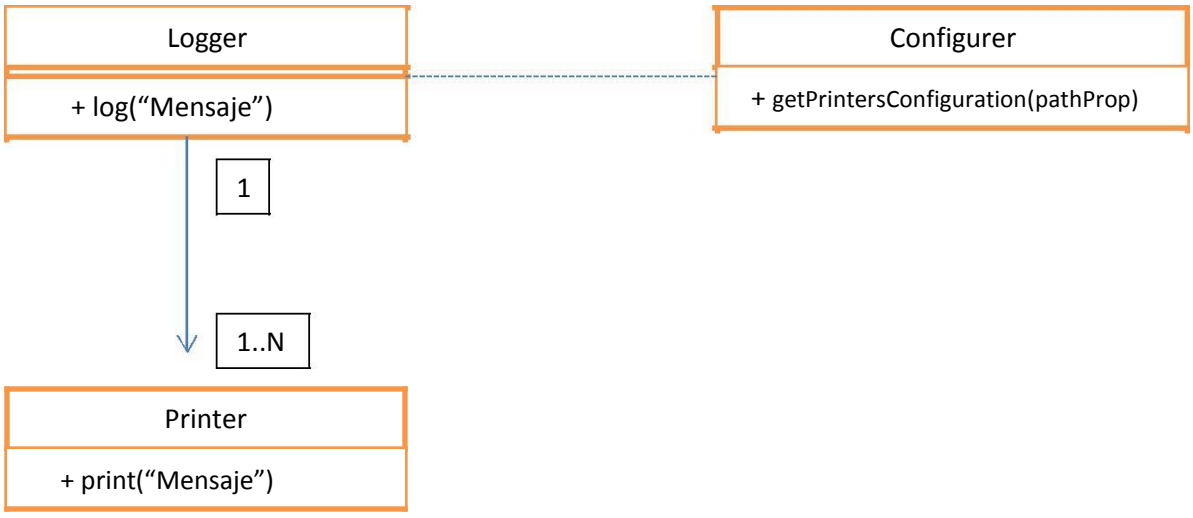
Logger.logError(“mensaje de prueba ERROR”);

Logger.logDebug(“mensaje de prueba DEBUG”);

NOTA IMPORTANTE: antes de finalizar el logger hay que llamar el método close para que se encargue de cerrar las impresoras abiertas

DISEÑO

Para mayor detalle sobre el diagrama de clases, ver el diagrama que se adjunta



La intención del grafico anterior es mostrar cuales son las 3 clases más importantes del diseño

Configurer: se encarga de acceder al archivo properties y parsear las configuración previamente seteadas para luego facilitarcelas a la clase Logger

Logger: proveer la api para realizar los logs

Printer: recibe el mensaje a loguear y le aplica el determinado formato, previamente definido en la clase Configurer

Diseño del configurer

Toda la información de configuración de las distintas impresoras esta almacenada en una estructura Properties, la misma es atributo de la clase Configurer

Dentro de la misma existen dos claves primordiales en el diseño de la aplicacion,
la clave “files” y la clave “consoles”

Ambas claves contienen los nombres de los distintos archivos o consolas generadas por el usuario concatenadas y separadas entre si por una “,”
Las características de cada impresora estan almacenadas mediante la siguiente formula de clave:valor(Explicacion con mayor detalle al comienzo de la documentaicon)

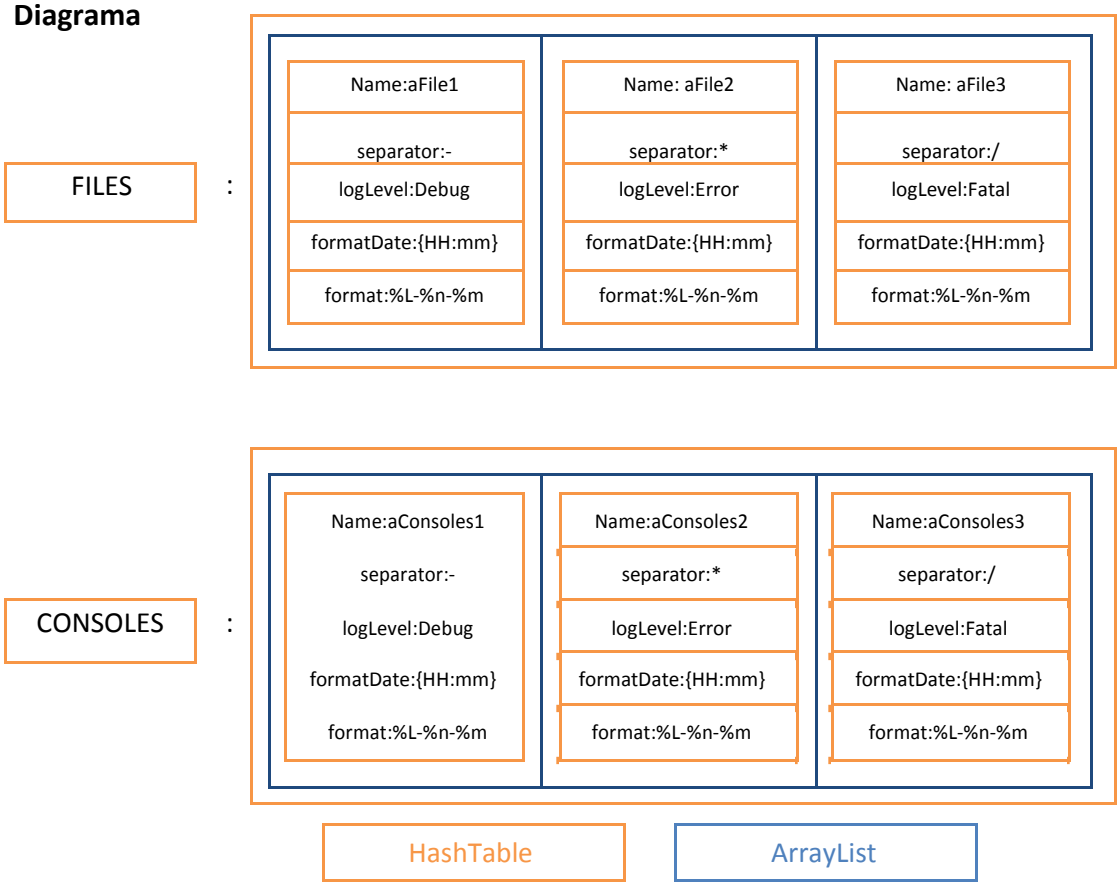
nombreImpresora-caracteristica = valor

Para pasar toda esta información a la clase Logger se usa una estructura HASHTABLE cuyas dos principales claves son FILES y CONSOLES, y cada una de estas claves apuntan a un ArrayList con contiene por cada nodo un HASHTABLE con la configuración de cada Impresora, a continuación se adjunta un diagrama para poder ampliar el panorama

Declaracion de la estructura en JAVA:

Hashtable< String , ArrayList < Hashtable < String, String > > >

Diagrama



Luego la clase Logger se encarga de parsear esta estructura y crear las impresoras

Diseño de la Clase Logger

En su instanciación recibe una lista de hashes(esta estructura esta detallada en la sección previa) con la información pertinente a cada impresora, esta clase se encarga de parsear dicha estructura y transformar cada nodo del hash que representa cada impresora en un instancia de la clase Printer, cada una de ellas es guardada es un ArrayList

Cuando llega un mensaje a loguear, este se pada a cada miembro de este ArrayList para que lo formatee y lo imprima en su respectiva salida.

En el caso de no poder loguear un mensaje debido a que no supera el nivel de logging, se escribe el error en el archivo error.dat, el mismo está en la carpeta raíz del proyecto

Diseño de la clase Printer, FilePrinter, ConsolePrinter y justificación de herencia

Básicamente se decidió usar herencia y no interface en este punto debido a que las clases que heredan de Printer, sean FilePrinter y ConsolePrinter, compartían ambas los mismos atributos, los mismos métodos, y la funcionalidad de ambas es exactamente la misma, salvo que una imprime en un archivo y la otra por consola, por lo que todo lo común se subio a la clase Printer y únicamente se dejó en cada clase la implementación de la impresión de los mensajes y cierre de impresora

Los archivos de log van a la carpeta log en la carpeta raíz del proyecto

Relacion de la Interface Pattern con la clase Printer y la clase FactoryPattern

La clase Printer posee un ArrayList de Patterns, dichos pattern son creados por la Clase FactoryPatterns, este recibe una lista de los patrones a aplicar al mensaje de cada impresora, osea cada instancia de ConsolePrinter o FilePrinter, parsea la lista y crea las dististas Pattenrs.

Pattern no es una clase, es un interface implementada por cada clase a la cual se le delego la aplicación de un determinado formato al mensaje(Ver Tema Formatos Mensaje en enunciado del TP)

Dichas clases son PatternDate, PatternEscape, PatternFileName, PatternLevel, PatternLineNumber, PatternMethodName, PatternSeparator, PatternSimpleMessage, PatternThread, PatternUserDefinedMessage

Lo que hace cada clase lo representa claramente su nombre, para mayor detalle ver comentarios en código Fuente de cada clase

NOTA: la clase PatternSimpleMessage fue un agregado de último momento, ya que se vio un mensaje en el foro que decía que se podían definir formatos de mensaje del estilo %L-%n-%d{HH:mm:ss}-%n-**HOLA**-%m , esta clase se encarga de tomar estas palabras aisladas como HOLA, en este caso, y concatenarlas al mensaje a imprimir.