75.10 – Técnicas de Diseño

TP 1

1º Cuat. 2014

Grupo 8

Integrantes:

90697 - Eisner, Ariel

89563 - Ferreyra, Federico

89992 - Cerrota, Matías

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

Uso de la API para configurar la aplicación

La encarga de llevar a cabo esta tarea es la clase Configurer.

Configuración por defecto

Al instanciar la misma se llama al método **`loadDefaultProperties()`** que se encarga de setear algunas variables por defecto, a saber:

- `Separador = "-"`
- `el nivel de Log="Debug"`
- `el formato de fecha ="%d{HH:mm:ss}"`
- `y un formato de mensaje = %d{HH:mm:ss}-%n-%p-%n-%m`

para poder arrancar con una forma de mensaje ya preestablecida

La impresión por default se hace por consola y el nombre de la misma es `"consoleDefault"` .

Luego de haber iniciado el Configurer, si ejecutamos el comando:

```
log.log("Mensaje  de Prueba",LevelLog.DEBUG);
```

Saldrá por consola :

```
"11:21:22-DEBUG- Mensaje  de Prueba"
```

Modificando las propiedades de la consola por default

Las propiedades que se pueden modificar para la salida por default(y para todas las que mas adelante veamos como se crean) son las que viene seteadas por defecto y se hacen a través de los métodos

```
setFormatDate(String Printer,String formatDate),

setFormatMessage(String Printer,String formatMessage),

setLogLevel(String Printer,String formatLevel),

setSeparator(String Printer,String formatSeparator)
```

El Segundo parametro representa el valor de la propiedad a modificar, y el primero es el nombre de la "Impresora ␣ Printer" sobre la cual se va a setear la propiedad, es este caso la consola se llama "`consoleDefault`" y por ejemplo para modificar el nivel de log nos manejaríamos de la siguiente manera:

```
setLogLevel(“consoleDefault”,”ERROR”)
```

y apartir de este momento solamente por la consola por defecto se pueden loguear mensajes del nivel ERROR o inferior.

Creando nuevas impresoras(Printers)

Se pueden crear tantas impresoras como queramos, las mismas forman parte de dos categorías CONSOLES Y FILES.

Como sus nombres lo indican la primer lista es para loguear a distintas consolas y la segunda es para hacerlo a distintos archivos

Para crear un Printer FILE hay que invocar al método

```
createPrinter(String categoriaPrinter,String namePrinter);
```

El primer parametro representa la categoria de la impresora a crear y el segundo el nombre de la misma, si es un archive, el nombre es igual que el nombre del archivo

Si quiero crear una impresora que loguee a un archivo llamado “log.dat”, debo llamar al método de la siguiente manera

```
createPrinter(“files”,”log.dat”);
```

Para crear otra consola es lo mismo pero cambiando la categoría

```
createPrinter(“consoles”,”otraConsola”);
```

Configurando la nueva impresora

Cuando se crea una nueva impresora la misma viene configurada con las mismas características que la consoleDefault, y para modificar la misma, sea creada en FILES o CONSOLES se usan los mismo métodos que se usan para configurar la consola por defecto. Por ejemplo, si quiero loguear a un archivo con level “INFO”, formato fecha “{HH:mm}” y separador “*”, la configuración la hago con la siguiente serie de comandos:

```
createPrinter(“files”,”unArchivo”);

setFormatDate(“unArchivo”, %d{HH:mm});

setLogLevel(“unArchivo”,”INFO”);

setSeparator(“unArchivo”,”*”);
```

y al momento invocar

```
log(“mensaje2 de prueba”,LogLevel.INFO);
```

se imprimirá en el archivo “unArchivo”

```
11:21*INFO*mensaje2 de prueba
```

Dando formato al mensaje de salida

Para dar formato a la salida de una determinada impresora debo llamar al método

```
setFormatMessage(“nombreImpresora”,”formatMessage”),
```

La variable “formatMessage” esta conformada de distintos campos que indicant la información que conformara cada línea del archivo de logueo, dichos campos deben ir separados por un “-” para que no haya problemas a la hora de parsear los mismo, ie:

```
setSeparator(“nombreImpresora”,”/”);

setFormatMessage(“nombreImpresora”,” %L-%n-%d{HH:mm:ss}-%n-%p-%n-%m”);
```

al loguear el mensaje “mensaje de prueba 3” se imprimira

```
3/11:21:22/DEBUG/mensaje  de prueba 3
```

NOTA: el primer 3 ser refiere a la línea número 3 de la salida de la impresora

Borrando la consola por default

Si quiero eliminar la consola por default debo llamar al método `eraseDefaultConsole()`.

Guardando la configuración establecida

Luego de crear las consolas o files a donde loguear, si quiero guardar la configuración debo llamar al método `saveProperties()`.

Integracion de la configuración a la herramienta de Logging

Una vez finalizada la creación y configuración de las distintas impresoras a utilizar, hay que informarle al “Logging” cuales 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 herramiento ya queda lista para podes utilizarla.

```
Logger loguer = new Logger( configurer.getPrintersConfiguration() );

loguer.log("Mensaje1",LogLevel.ERROR);
```

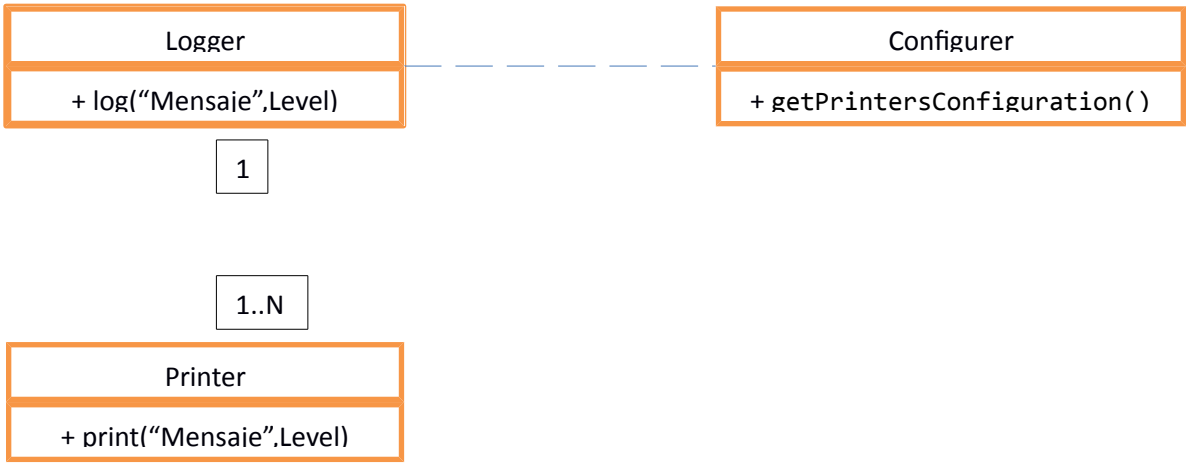
Uso de la API de Logger

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

```
Logger.log(“mensaje de prueba”,LevelLog.INFO);
```

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 mas importantes del diseño

Configurer: se encarga de la configuración de la aplicación

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 de la clase Printer

La misma esta explicada debajo, donde también se justifica el uso de herencia en la misma .

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 la solución, la clave "files" y la clave "consoles"

Ambas claves contienen los nombres de los distintos archivos o consolas generadas(Ver uso de Api Configurer) por el usuario concatenadas y separadas entre si por una ","

Las características de cada impresora esta almacenada mediante la siguiente forma clave:valor

nombreImpresora-caracteristica=valor

por ejemplo, para una impresora de tipo files con nombre “log.txt”, las caracterisitcas seteadas por el usuario se almacenan así:

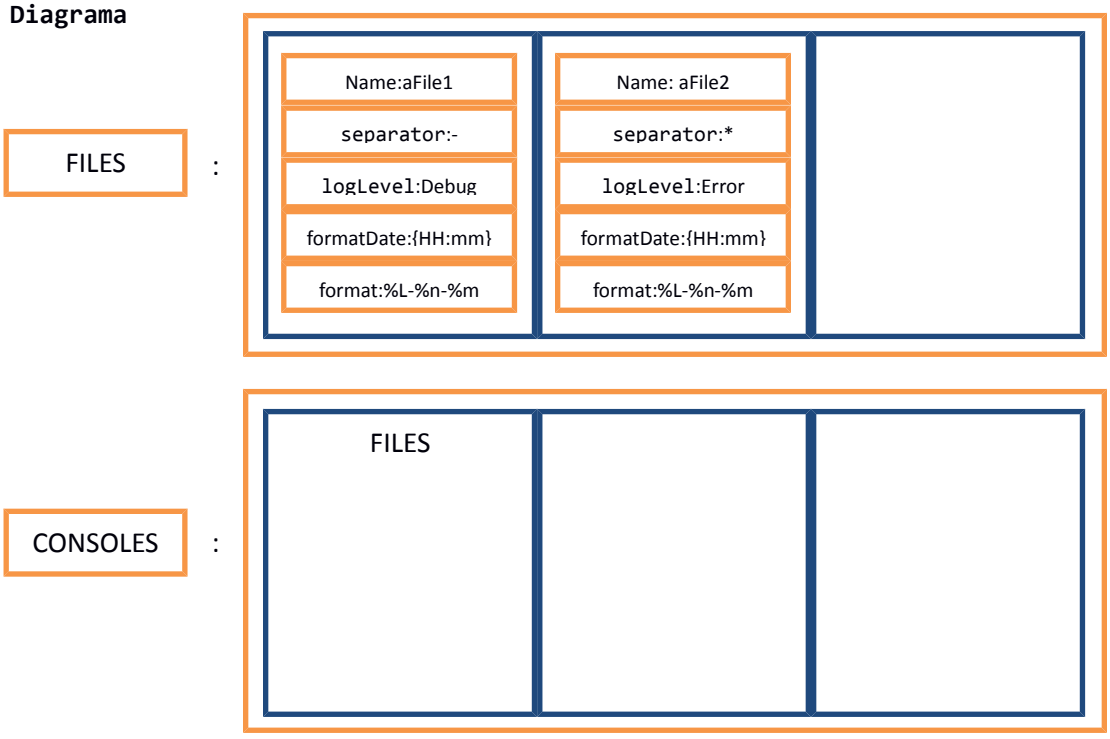
```
log.txt-separator=-

log.txt-logLevel=DEBUG

log.txt-formatDate="%d{HH:mm:ss}

log.txt-format =-%d{HH:mm:ss}-%n-%p-%n-%m
```

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.

<u>Diseño de la Clase Logger</u>

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 imprima en su respectiva salida.

Diseño de la clase Printer y justificación de herencia

Basicamente se decidio 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 dejo en cada clase la implementación de la impresión de los mensajes

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 comentacion en codigo fuente de cada clase

NOTA: la clase PatternSimpleMessage fue un agregado de ultimo momento, ya que se vio un mensaje en el foro que decie que se podian definer 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 esta caso y concatenarlas el mensaje a imprimir.