TP1 - TÉCNICAS DE DISEÑO - LOGGER

GRUPO N° 22
- *Marcos Forlenza (87237)*
- *Belen Beltran (91718)*
- *Fiona Gonzalez Lisella (91454)*

RESUMEN:
La siguiente api presenta un cliente de trazas simple. En el presente documento se comentarán brevemente decisiones de diseño, y casos particulares de resolución.

DISEÑO:
Como parte de la investigación inicial del desarrollo se estudiaron los fuentes de dos loggers conocidos. Por una parte log4j y también la implementación nativa de Java.
También se observó el funcionamiento del Logger de Eclipse ADT al estar conectado a algún dispositivo.

El estudio se tornó un poco complejo en cierto punto, pero sirvió simplemente para entender algunas abstracciones útiles como el Appender y el LoggerManager.

Con estos ejemplos en mente, se procedió a diseñar el Logger presentado.
En este diseño, se utilizaron (sin hacerlo de forma estricta) patrones vistos en clase como el Factory, Builder, Singleton.
Básicamente los loggers se almacenan en un mapa y para obtener cada uno se utiliza un llamado estático (Logger.getLogger("Clase")) como se acostumbra a utilizar en todos los loggers conocidos. Estos loggers son administrados por una entidad llamada LoggerManager que está delegada en la clase principal Logger. Logger también tiene referencias a un objeto de configuración que encapsula las configuraciones cargadas de un archivo. Para realizar la creación se utilizaron dos objetos constructores un LoggerFactory y un MockFactory. El primero lee un archivo de configuración utilizando un objeto Properties y el segundo tiene datos fijos utilizados para las pruebas
El formato de los mensajes se obtiene a partir de un String que se lee del archivo de Properties. Éste permite agregar varias opciones al mensaje original, como por ejemplo la fecha, el nivel, etc. Permite también realizar varias combinaciones entre las opciones elegidas. Un ejemplo de formato podría ser: "%d{HH:mm:ss} %p %t %m".

PUNTOS FUERTES Y DÉBILES DE LA SOLUCIÓN

La solución se implementó de la forma más simple posible. La idea fue que sea fácil de entender, que se pueda explicar por sí sola. Se tuvieron en cuenta los resultados de la investigación previa para que la interfaz de entrada, es decir la forma principal de loguear, sea similar a la de las api que circulan. De esta forma el usuario puede acostumbrarse rápidamente al uso y al mismo tiempo puede contar con una solución más liviana y con menos dependencias.

Otro punto fuerte de la solución es que la creación de un logger se encuentra sumamente encapsulada y utilizando patrones de creación que hacen sencillo realizar distintos test.

También debemos resaltar la idea de “appenders” que son todos aquellos objetos que implementen la interfaz LoggerAppender y que son los distintos destinos de las trazas. Es sencillo adaptar un nuevo destino de log, como podría ser una base de datos, simplemente generando un nuevo objeto que implemente esta interfaz y delegando la funcionalidad específica a este mismo objeto.

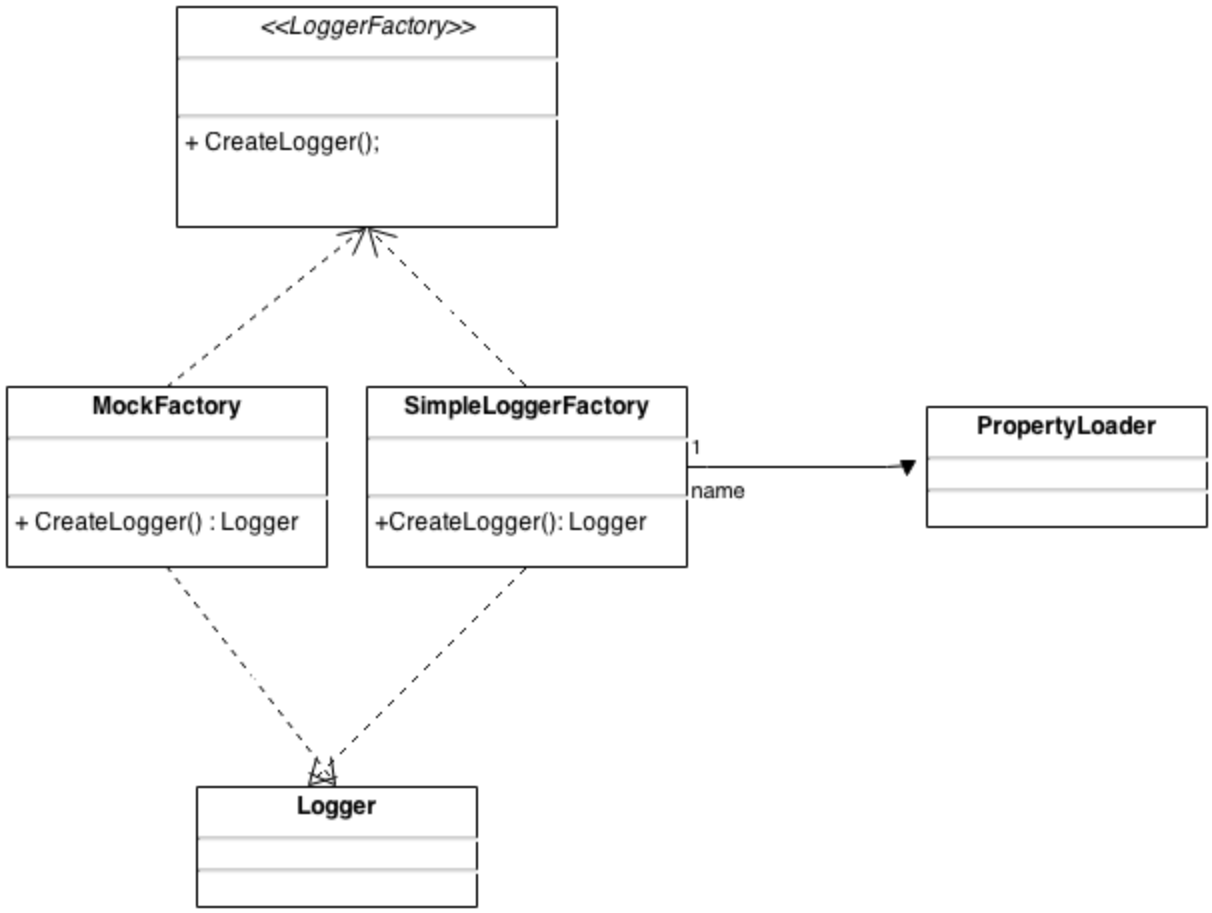
En el diseño siempre se tuvo en cuenta la segregación de interfaces, la extensibilidad y las posibilidades de testear de forma unitaria.
Se puede destacar como un punto débil de la solución la cantidad de test que forman el código. Si bien el código fue pensado para ser probado no se diseñaron una gran cantidad de test. La idea es ir completando los mismos pero lo mejor hubiese sido realizarlos antes de construir el código.

Por otro lado se puede destacar como un punto débil la configuración estándar por archivo properties. Existieron distintas ideas para modificarlo (xml, annotations, json) que no fueron implementadas por razones de tiempo.

DIAGRAMA DE CLASES:
Se adjunta en los fuentes los archivos con los diagramas de clases generales y específicos a la resolución del problema. Los mismos se encuentran en el subdirectorio */doc*.

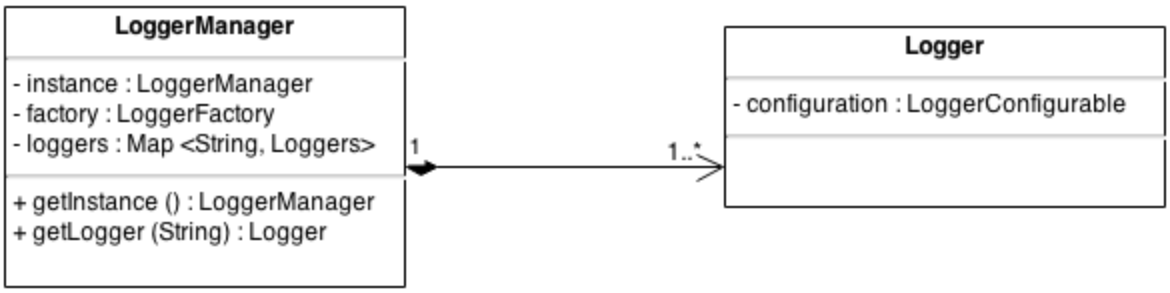
PATRONES Y CLASES RELACIONADAS:

Patrón Factory
Permite crear objetos a partir de distintas fuentes de información, ocultando la interfaz del constructor del objeto al cliente. En este caso, se utiliza para construir el logger a partir de un archivo, pero podria ser fácilmente extensible a otras fuentes donde se encuentre la información para la configuración (ej: formulario, consola, archivo xml, etc).



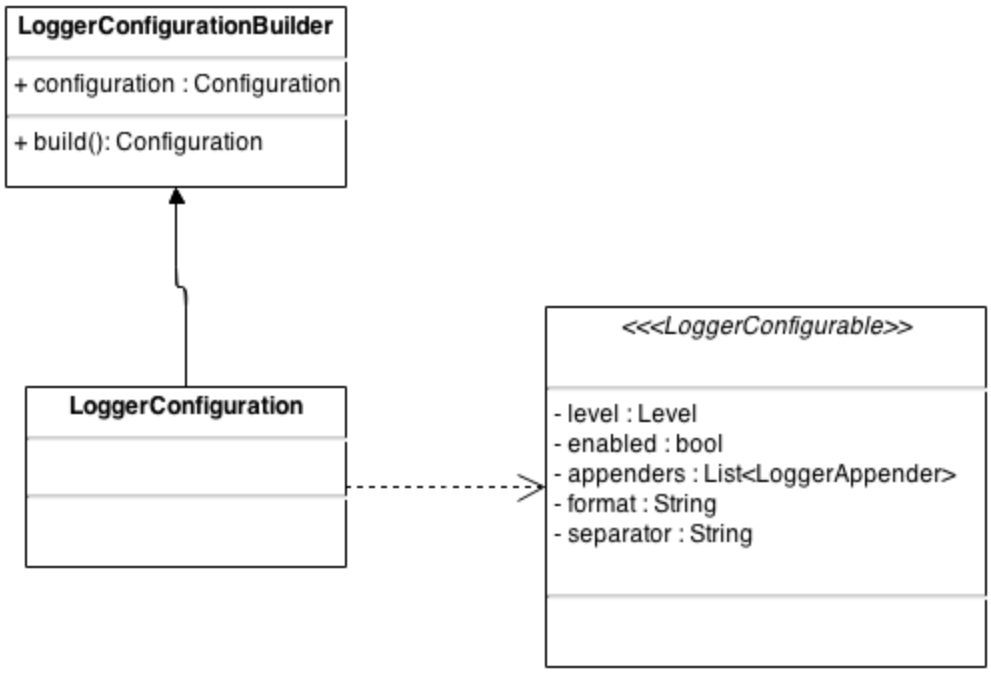
Patrón Manager

Permite administrar las instancias de logger, devolviendo la instancia correcta del logger que se solicita. Evita que cada logger por sí mismo sea un singleton.



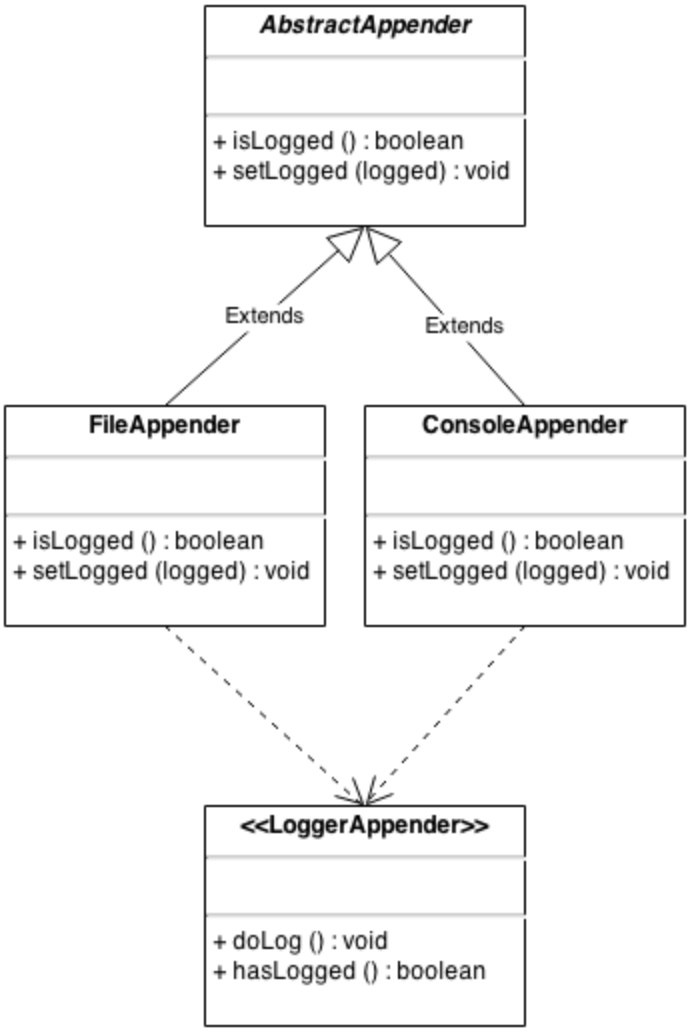
Patrón Builder

Permite encapsular la construcción de la configuración del logger, dejándola en un estado válido.



Diseño de Appenders

El siguiente diagrama representa la organización de clases para el manejo de appenders en la resolución del problema.
El diseño es facilmente extensible dado que para agregar nuevos appenders, como por ejemplo, texto html, stream, base datos, envio de mails, etc.



DECISIONES TOMADAS:
Para configurar los appenders en el archivo properties se tomó la siguiente convención:
- Cada File Appender tiene el siguiente formato:
logger.appender.file.NOMBRE_APPENDER=RUTA ARCHIVO
- Si se decide utilizar el Console Appender (loguear por consola) se ingresa la siguiente línea:
logger.appender.console = true
- Para indicar el separador a utilizar se utiliza la siguiente línea:
logger.message.separator = “SIMBOLO”
- Para utilizar un formato pre establecido se utiliza la siguiente forma:
logger.message.format = “FORMATO PREESTABLECIDO”

Para formatear el mensaje se separada la línea del formato utilizando el separador configurado que no puede ser los caracteres utilizados para el formateo, no pueden ser letras ni los signos %, {, } :
El orden con que se escriba la línea será el orden del mensaje a trazar.
Los códigos de formato son los siguientes:
● %d{xxxxxxx} -->Cualquier formato válido de SimpleDateFormat.
● %p --> Nivel del mensaje.
● %t --> Nombre del thread actual.
● %m --> Contenido del mensaje logueado por el usuario.
default a elección.
● %L --> Line number.
● %F --> Filename.
● %M --> Method name.

A REALIZAR A FUTURO:
En principio se habían pensado este modificador extra:
● %% debería mostrar un % (es el escape de %).

Para próximas versiones se podría cambiar la funcionalidad y permitir el escape del caracter %.

2da Entrega
CAMBIOS REALIZADOS:

- Se agregan los test unitarios sugeridos en la corrección. Si bien no hacen una gran diferencia en cantidad testean el método principal de log
- Se agrega una clase utilitaria que realiza comparaciones para facilitar el equals entre LoggerAppenders
- Se agrega un MockAppender para realizar testeos de forma más sencilla
- Se incorpora a la configuración el formato del logger y deja de ser creado por fuera de la misma.

- Se agrega un documento con descripción más detallada de la solución y diagramas de los patrones utilizados. También se agrega un análisis de los puntos fuertes y débiles de diseño.
- Para evitar el código duplicado en MessageFormat, se agregó un método *find* que encapsula la búsqueda del modifier en el string del formato.
- Se logró que los tests sean más legibles a simple vista.
- Se agregaron setups en los tests
- Se cambia de paquetes aquellas entidades de Mock
- Se cambió la comparación de entero y se agregó una excepción que detalla el caso de error en el formateo de mensajes.