Ejemplos de uso

Configuración default

El logger se puede usar "out of the box" con las configuraciones default, de la siguiente manera.

```
Logger loggerInstance = LoggerImpl.getLogger();
loggerInstance.logMessage("This is a log", LogLevel.LEVEL_DEBUG);
```

Configuración custom

Formato de mensaje

```
LogFormat messageFormat = new LogFormatImpl("%p - %m");
loggerInstance.setMessageFormat(messageFormat);
```

Nivel de debugging

```
loggerInstance.setLogLevel(LogLevel.LEVEL_INFO);
```

Output por consola

```
loggerInstance.setConsoleOutput(true);
```

Agregar un archivo de output

```
loggerInstance.addOutputFile("path/to/output");
```

Múltiples loggers

Se agregó el soporte de múltiples loggers, que pueden ser utilizados de la siguiente manera.

```
Logger loggerInstance1 = LoggerImpl.getLogger("logger1");
Logger loggerInstance2 = LoggerImpl.getLogger("logger2");
Logger loggerInstance3 = LoggerImpl.getLogger("logger3");

loggerInstance3.setLogLevel(LogLevel.LEVEL_OFF)

loggerInstance.logMessage1("This is the first log.",  LogLevel.LEVEL_DEBUG);
loggerInstance.logMessage2("This is the second log.", LogLevel.LEVEL_TRACE);
loggerInstance.logMessage3("This won't get logged.",  LogLevel.LEVEL_TRACE);
```

Destinos custom

El usuario del logger puede utilizar un destino Custom, implementando la interfaz Writer. Para archivos y consola, se implementaron las clases FileWriter y ConsoleWriter respectivamente. Las misas sirven como ejemplo para crear otros destinos.

Decisiones tomadas

- Delegar la comparación entre los distintos tipos de Log Levels para evitar tener que modificar distintas clases si esta jerarquía sufre una modificación en su especificación. Se redujo el acoplamiento entre las clases.
- Variables de clase en LogLevel, para un facil acceso a instancias de LogLevel, utilizadas para comparar y para pasar como parámetro al método logMessage.

- La utilización de filtros para el formato de mensajes. De esta manera, cada filtro se encarga de una porción del string a reemplazar, pudiendo encadenar varios filtros sucesivamente, generando el log con los campos interpolados.
- Utilización de clase abstracta de los filtros, para reutilizar código que se repite. Por ejemplo, cuando el filtro es genérico, y se reemplaza un string por otro.
- Se definieron excepciones custom, para obtener un mayor control y especificidad en los posibles errores.
- Uso de delegate pattern para realizar la lectura del archivo de properties. El logger es delegate de un cargador de propiedades, que lo conoce a través de la interfaz PropertyApplyingDelegate.
- Interfaz para sellar el código ante cambios a la hora de realizar la escritura en archivos/consola. Se utilizó una interfaz Writer, que deberá ser implementada por cualquier tipo de output al que se pretenda loggear.

Impacto generado por nuevos requerimientos

- Se decidió abstraer la manera en la que se hace el log en distintas estrategias (Strategy) para poder soportar los 2 tipos de formatos (Strings y JSON).
- El nuevo nivel de logging fue sencillo de implementar, ya que solo se debió tocar a la jerarquía de log levels.
- Los distintos loggers en simultáneo también fueron sencillos de implementar, ya que a nuestro método que sobrecargamos el método que obtiene el singleton para que pueda recibir un nombre y agregamos un diccionario de loggers que hacen las veces de singletons.
- Los filtros configurados por el usuario también fueron sencillos de implementar, ya que la toma de decisión de si un mensaje debía ser logueado estaba en un solo lugar. Fue necesario simplemente agregar una condición en ese lugar.
- Para soportar la nueva pseudo-variable, bastó con agregar un nuevo Filtro (así le llamamos a lo que interpola el string) para poder manejar el %g. Nuevamente, el reemplazo de estas variables estaba solo en una clase, por lo que fue sencillo identificar el código a modificar.
- El último de los requerimientos, los destinos genéricos, fueron soportados gracias a una corrección por parte de nuestro ayudante, que nos remarcó que nuestro código no estaba clausurado ante cambios de la manera que manejamos la escritura a archivos. Al abstraer este mecanismo, naturalmente surgió una interfaz que puede ser implementada por un usuario para determinar un destino custom de logging.
- Se decidió generar nuevos niveles de paquetes, para favorecer el rápido acceso a código y la alta cohesión dentro de los mismos.

Patrones de diseño utilizados

- **Pipe and Filter:** utilizado en la interpolación de las variables a la hora de darle formato a un mensaje. Los filtros se van encadenando uno después del otro para obtener el string final con todas las variables interpoladas.
- **Singleton:** las instancias de los loggers son tratadas como singletons, ya que un logger previamente configurado debe poder ser utilizado en cualquier lugar de la aplicación cliente.
- **Strategy:** se utilizaron distintas estrategias a la hora de darle formato a un mensaje. Las dos estrategias que utilizamos para convertir a String fue la de interpolar la cadena de formato y la de convertir un objeto contenedor a JSON.
- **Adapter:** se eligió implementar un adaptador para poder utilizar nuestro logger como back end de la libreria SL4J.
- **Factory Method:** se utilizó una fábrica para poder crear los distintos log levels a partir de un string. También se utilizó para crear los distintos lectores de configuración.