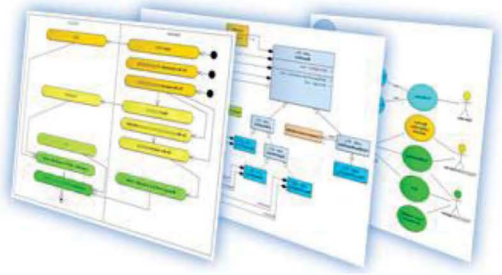
Tema 2: Entornos de Desarrollo - Logger

Tema 2: Entornos de Desarrollo
Entornos de Desarrollo



Alejandro Cardo Grau

T2: Entornos de Desarrollo

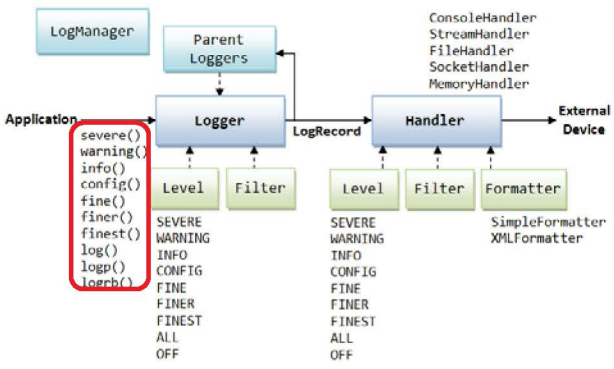
5. Manejo de Herramientas para la Depuración.

- **Java Logging API**
  - **API de Java para la emisión eficiente de logs: configurable mediante código o con ficheros**
  - Opción no eficiente ni parametrizable:
    - Imprimir salida para todo sin nivelar el tipo de log

```
System.out.println("usuario entrante: user=" + x);
```

    - Además, se mezclan en consola todos los "*logs*" sin diferenciarlo:
      - En los IDE existen técnicas de filtrado por expresiones regulares que permiten diferenciar las cadenas y establecer estilos: colores, fuentes,etc
  - **Opción eficiente y parametrizable**

```
logger.warning("usuario entrante: user=" + x);
```



Alejandro Cardo Grau

- **Java Logging API**

java.util.logging

Class Logger

java.lang.Object
    java.util.logging.Logger

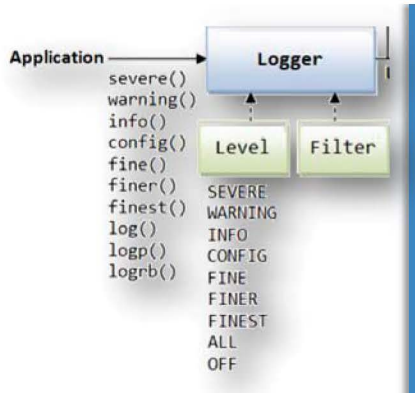
public class **Logger**
extends Object

A Logger object is used to log messages for a specific system or application component. Loggers are normally named, using a hierarchical dot-separated namespace. Logger names can be arbitrary strings, but they should normally be based on the package name or class name of the logged component, such as java.net or javax.swing. In addition it is possible to create "anonymous" Loggers that are not stored in the Logger namespace.

Logger objects may be obtained by calls on one of the getLogger factory methods. These will either create a new Logger or return a suitable existing Logger. It is important to note that the Logger returned by one of the getLogger factory methods may be garbage collected at any time if a strong reference to the Logger is not kept.

Logging messages will be forwarded to registered Handler objects, which can forward the messages to a variety of destinations, including consoles, files, OS logs, etc.

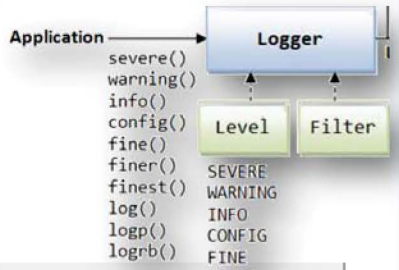
Documentación Clase Logger:
https://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html



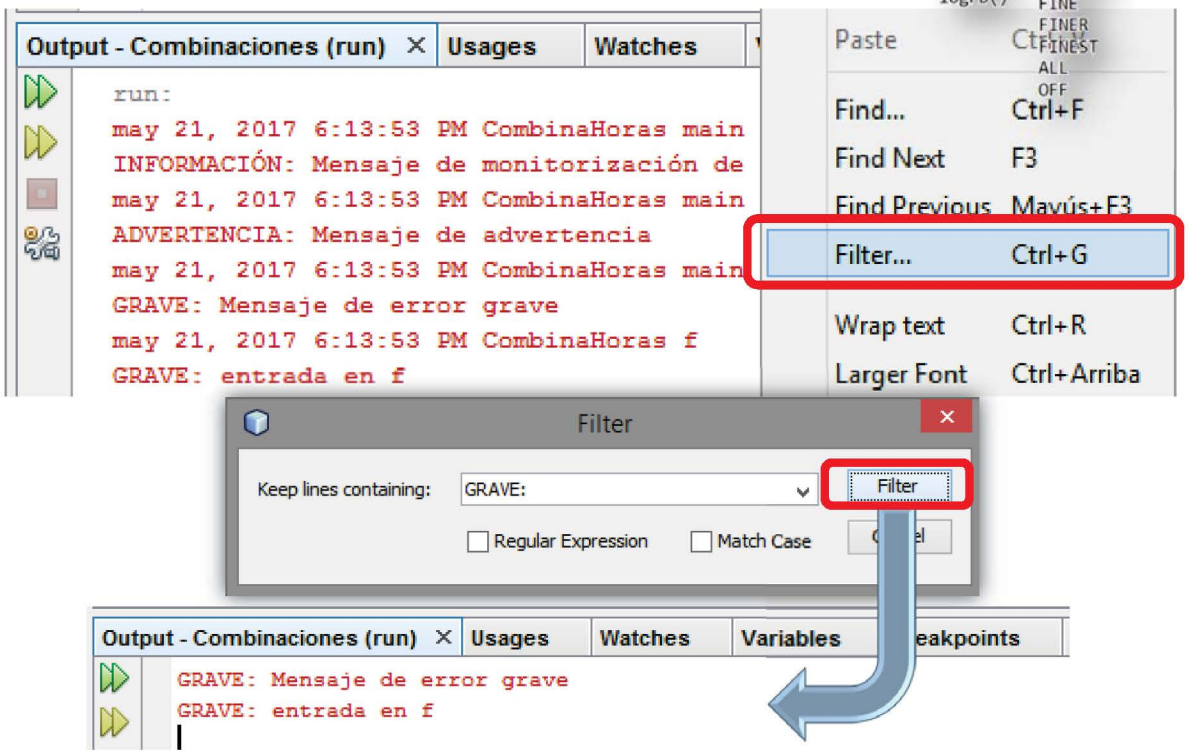
- **IDE Netbeans:**
  - **Definición y uso con Logger**

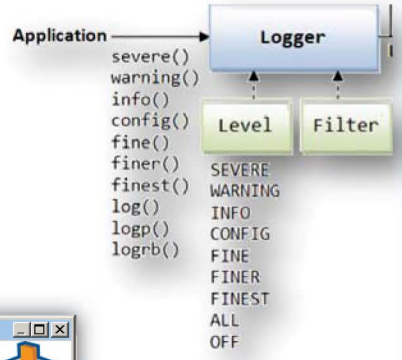
```
import java.util.logging.Logger;

public class CombinaHoras {

    private static final Logger logTag = Logger.getLogger(CombinaHoras.class.getName());

    public static void main(String args[]) {
        logTag.fine("Mensaje de depuración");
        logTag.info("Mensaje de monitorización de funcionamiento");
        logTag.warning("Mensaje de advertencia");
        logTag.severe("Mensaje de error grave");
        f();
    }

    public static void f() {
        logTag.severe("entrada en f");
    }
```

```
Output - Combinaciones (run) | Usages | Watches | Variables | Breakpoints
run:
may 21, 2017 6:13:53 PM CombinaHoras main
INFORMACIÓN: Mensaje de monitorización de funcionamiento
may 21, 2017 6:13:53 PM CombinaHoras main
ADVERTENCIA: Mensaje de advertencia
may 21, 2017 6:13:53 PM CombinaHoras main
GRAVE: Mensaje de error grave
may 21, 2017 6:13:53 PM CombinaHoras f
GRAVE: entrada en f
```

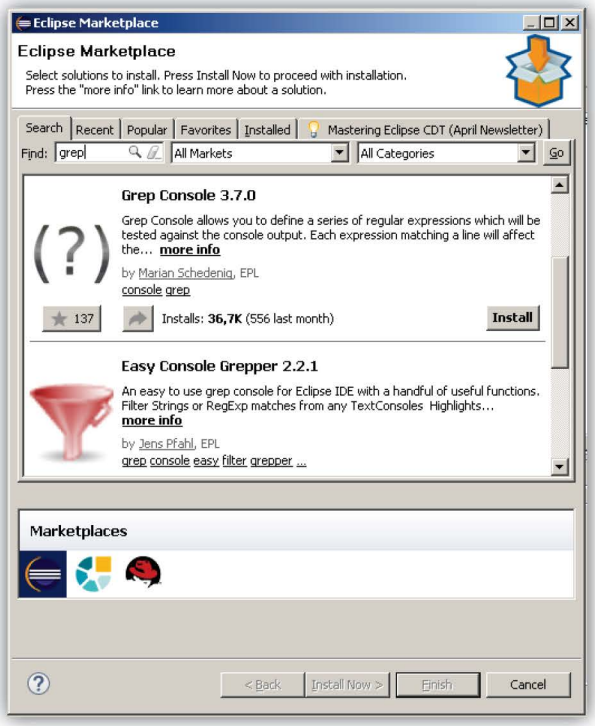


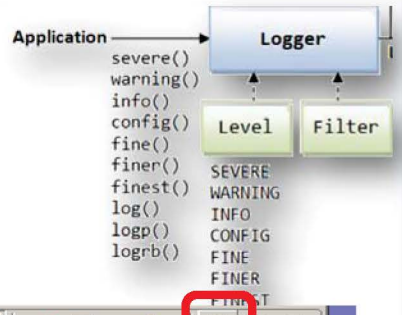
- **IDE Netbeans:**
  - **Output: Filtros de expresiones**



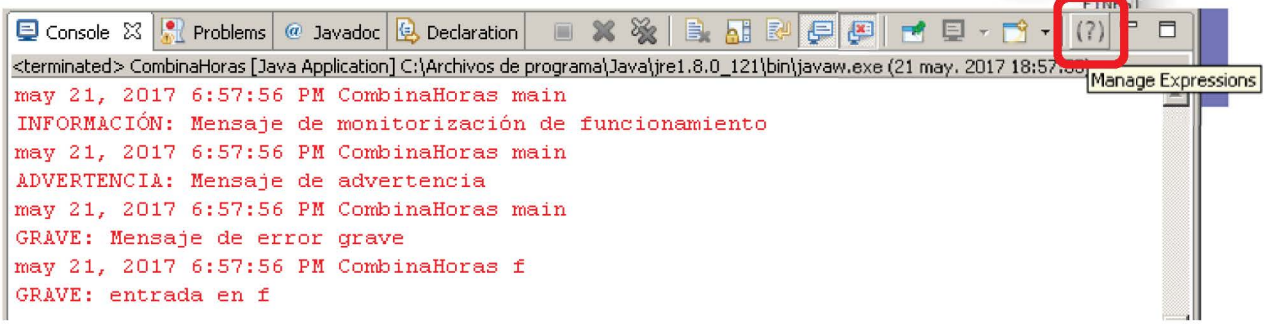


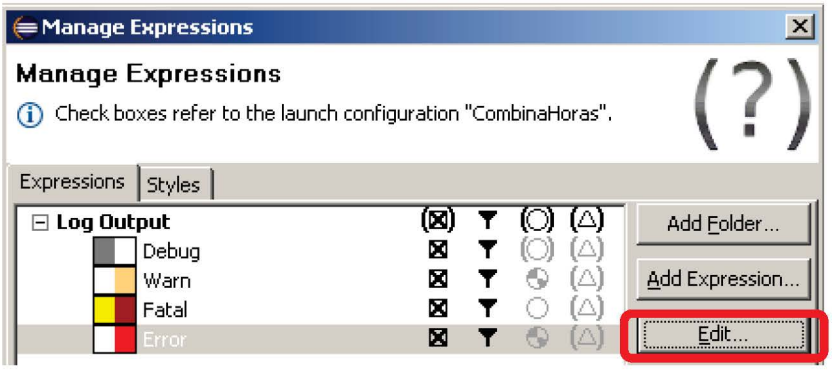
- **IDE Eclipse:**
  - **Console: Filtros de expresiones**
    - Instalación del plugin: *Grep Console*

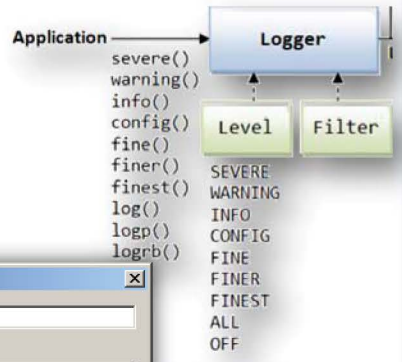




- **IDE Eclipse:**
  - **Console: Filtros de expresiones**

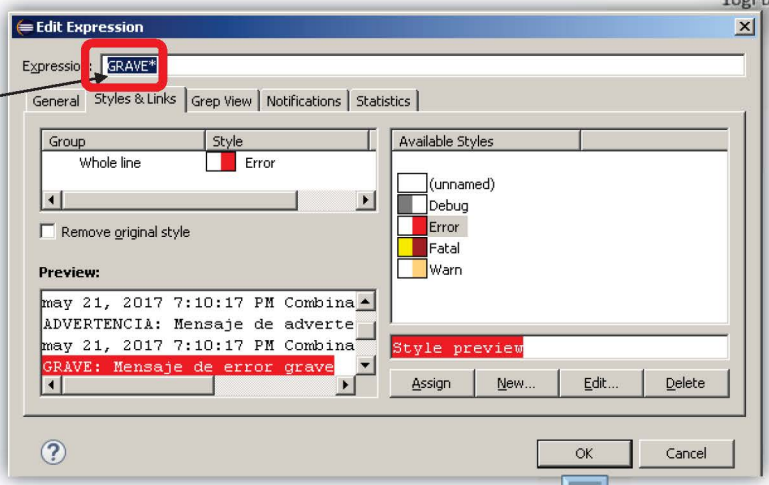


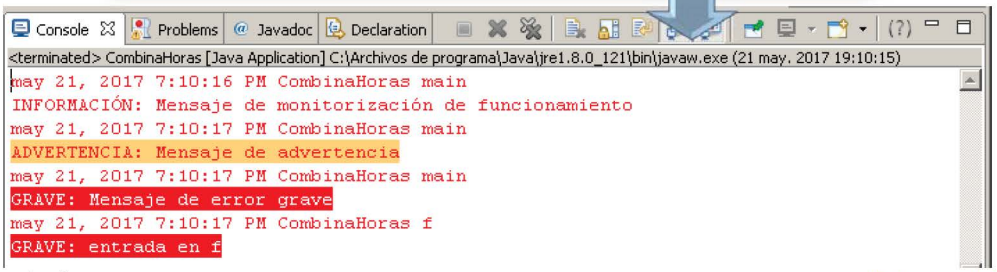




- **IDE Eclipse:**
  - **Console: Filtros de expresiones**

Introducir cadena de expresión para personalizar el filtro





Diferentes tipos de filtros personalizados según el nivel de *Logger*