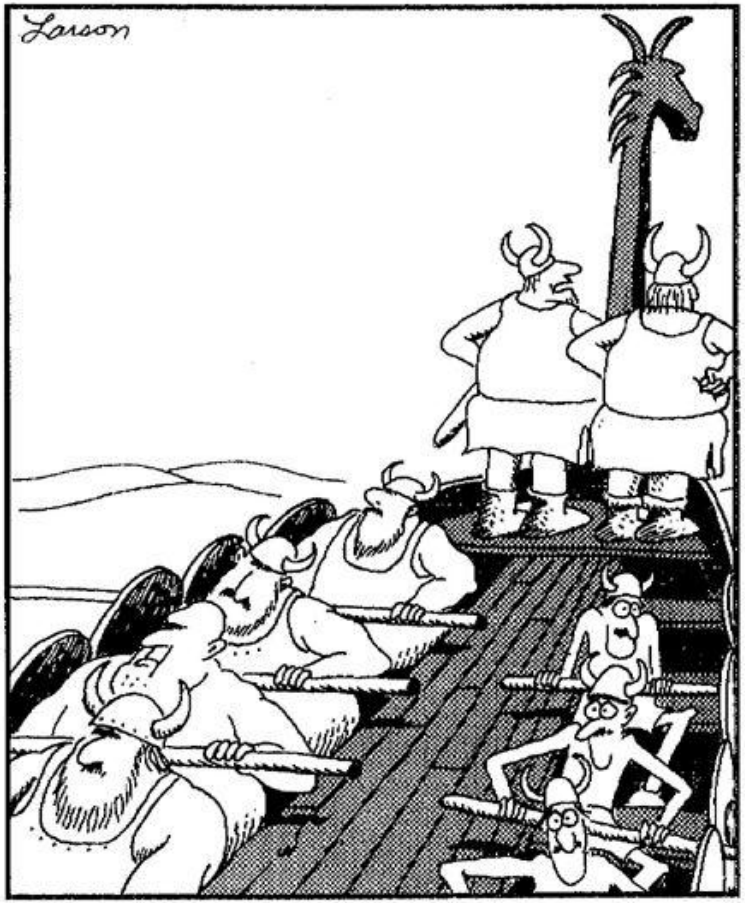


"I've got it, too, Omar ... a strange feeling
like we've just been going in circles."

Reuso: Framework II (blackbox)

Federico Balaguer

Resumen de la clase anterior

Frameworks:

- Proveen una solución reusable para una familia de aplicaciones
- Las clases en el framework se relacionan (herencia, conocimiento, envío de mensajes) de manera que resuelven la mayor parte del problema en cuestión
- El código del framework controla/usa al código de la instanciación
- Tipos de Frameworks
  - Aplicación: desktop, webapps, tcpservers :-)
  - Manejo Datos: ORDB, pipelines, NRDB
  - Sistemas Distribuidos: mensajes, eventos, rpc
  - Testing: unit, web pages
- Framework de Caja Blanca: las instanciaciones “completan” el loop de control agregando código
  - Ejercitando un hotspot con herencia
  - Modificando código fuente del framework

Dominio: TCP Servers

SingleThreadTCPServer (whitebox)

Cookbook:

1. Subclasificar SimpleThreadTCPServer
    a. Debe implementar Main(String[])
        i. crear una instancia
        ii. enviar método startLoop(String[])
    b. Debe implementar handleMessage(String) ⇒ hook

tcp.server.reply (blackbox)

Cookbook

1. En un objeto “contexto”
    a. instanciar un MessageHandler
        i. Echo,
        ii. Void
    b. Instanciar ConnectionHandler con el MessageHandler
        i. SimpleConnectionHandler
        ii. MultiConnectionHandler
    c. Instanciar TCPControlLoop con ConnectionHandler
    d. Enviar método startLoop() al TCPControlLoop

```java
import java.io.PrintWriter;

public class EchoServer extends SingleThreadTCPServer {

    public void handleMessage(String message, PrintWriter out) {
        out.println(message);
    }

    Run | Debug
    public static void main(String[] args) {

        new EchoServer().startLoop(args);

    }
}
```

SingleThreadTCPServer
(hotspot herencia)

```java
import tcp.server.reply.*;

public class EchoApp  {

    Run | Debug
    public static void main(String[] args) {

        new TCPControlLoop(new SingleConnectionHandler(new EchoHandler())).startLoop(args);

    }
}
```

tcp.server.reply
(hotspot composición)

```java
import tcp.server.reply.*;

public class MultiEchoApp  {

    Run | Debug
    public static void main(String[] args) {

        new TCPControlLoop(new MultiConnectionHandler(new EchoHandler())).startLoop(args);
    }
}
```

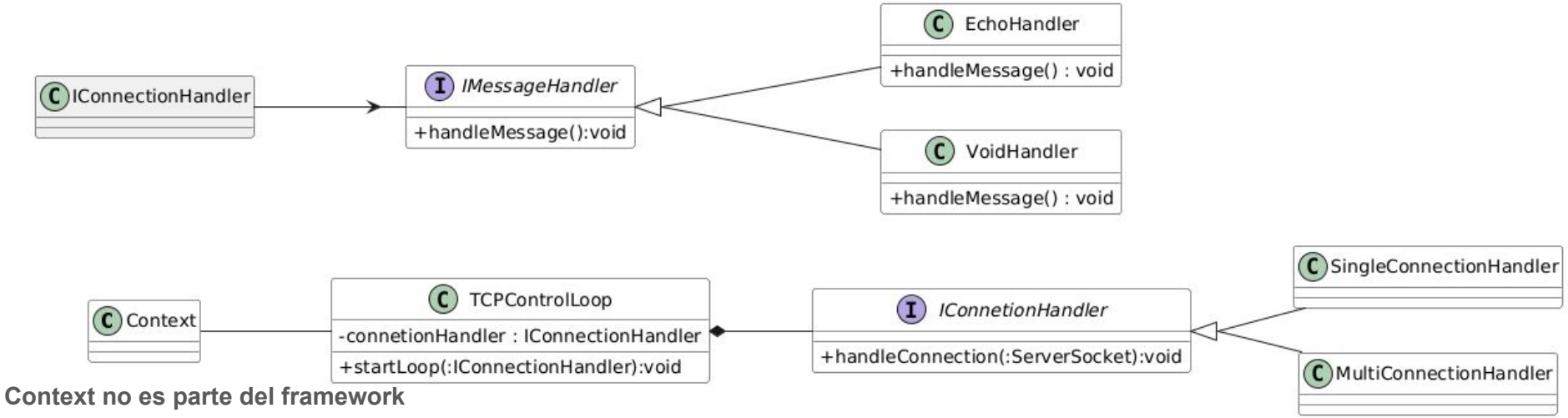
tcp.server.reply
multisession, echo handler

```java
import tcp.server.reply.*;

public class TestApp  {

    Run | Debug
    public static void main(String[] args) {

        new TCPControlLoop().startLoop(args);

    }
}
```

tcp.server.reply
singlesession, void handler

Dependency injection

tcp.server.reply



Context no es parte del framework

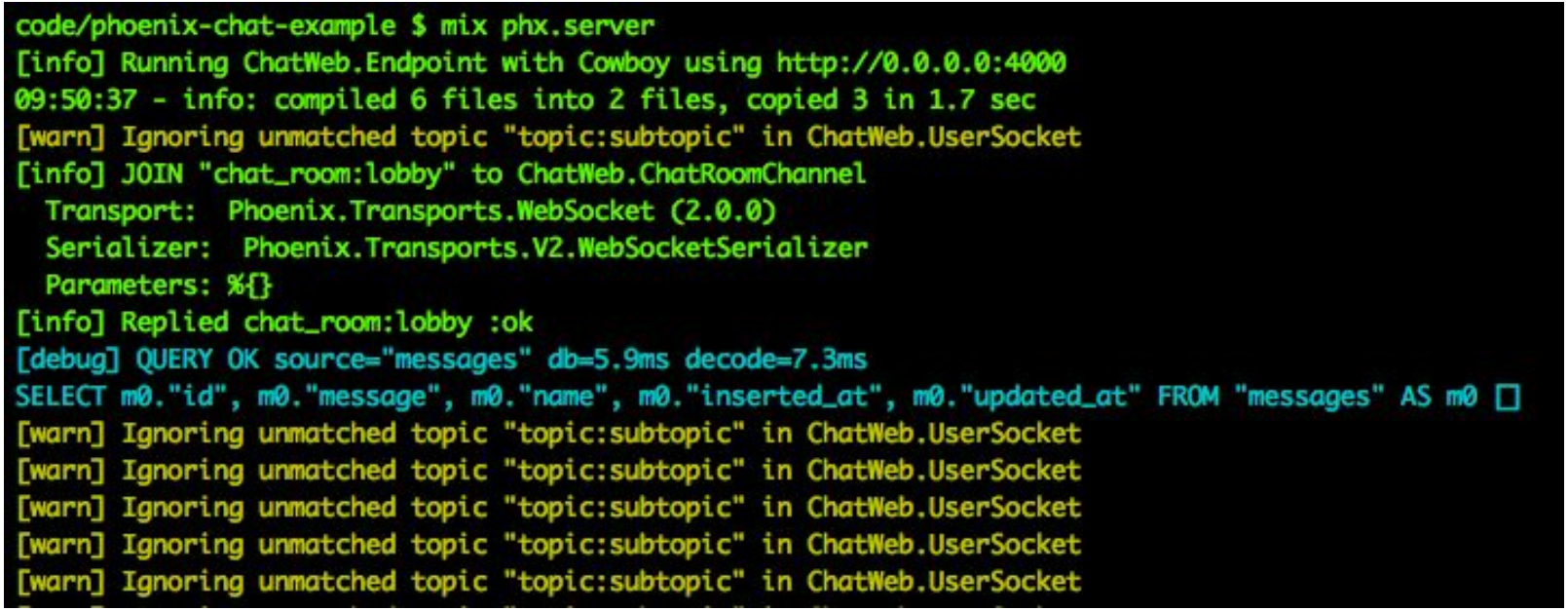
FrozenSpot:
Condición de corte de la conección
Un tipo de conexión (runtime)
Un tipo de MessageHandler (runtime)

HotSpot:
Nuevos MessageHandlers s/funcionalidad
Hash, timestamp, etc
Nuevos ConnectionHandlers
Timeout, recording, etc

tcp.server.reply: wrap up

- TCPControlLoop se configura con TCPConnection y MessageHandler
- El contexto puede ser:
  - servidor
  - parte de una aplicación
- Posibles mejoras:

1. Crear una Superclase de SingleConnectionHandler y MultiConnectionHandler
2. Crear la jerarquía de EndSessionPolicy
3. Modelar el concepto de Session
   a. En SingleConnectionHandler la sesión es el loop que procesa mensajes
   b. En MultiConnectionHandler la sesión es el TCPWorker (subclase de Thread)

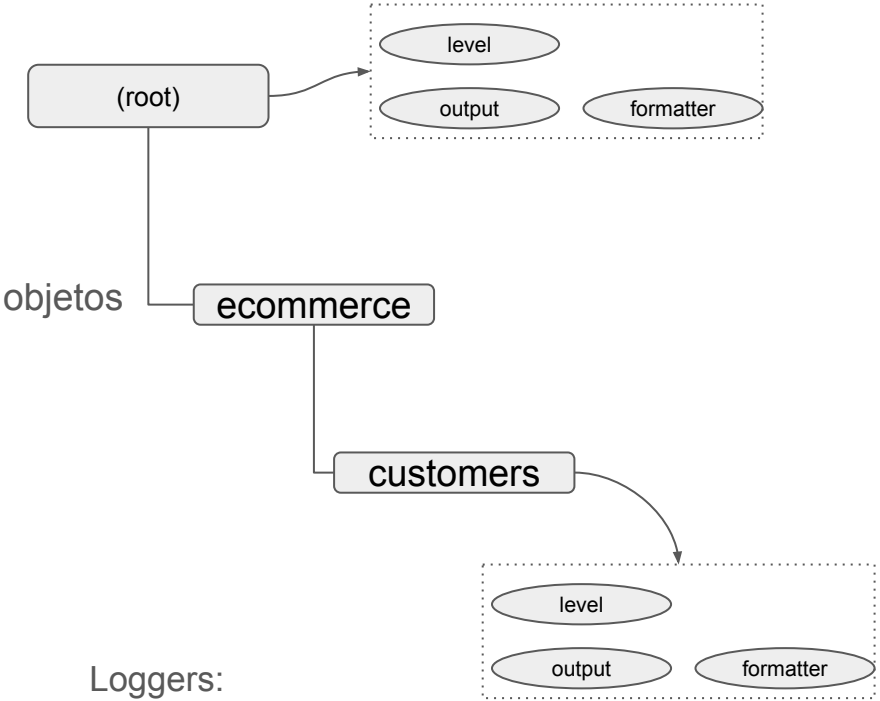

code/phoenix-chat-example $ mix phx.server
[info] Running ChatWeb.Endpoint with Cowboy using http://0.0.0.0:4000
09:50:37 - info: compiled 6 files into 2 files, copied 3 in 1.7 sec
[warn] Ignoring unmatched topic "topic:subtopic" in ChatWeb.UserSocket
[info] JOIN "chat_room:lobby" to ChatWeb.ChatRoomChannel
Transport: Phoenix.Transports.WebSocket (2.0.0)
Serializer: Phoenix.Transports.V2.WebSocketSerializer
Parameters: %{}
[info] Replied chat_room:lobby :ok
[debug] QUERY OK source="messages" db=5.9ms decode=7.3ms
SELECT m0."id", m0."message", m0."name", m0."inserted_at", m0."updated_at" FROM "messages" AS m0
[warn] Ignoring unmatched topic "topic:subtopic" in ChatWeb.UserSocket
[warn] Ignoring unmatched topic "topic:subtopic" in ChatWeb.UserSocket
[warn] Ignoring unmatched topic "topic:subtopic" in ChatWeb.UserSocket
[warn] Ignoring unmatched topic "topic:subtopic" in ChatWeb.UserSocket
[warn] Ignoring unmatched topic "topic:subtopic" in ChatWeb.UserSocket

java.util.logging

- Agregamos código de loggin a nuestra aplicación para entender lo que pasa con ella, por ejemplo:
  - Reportes de eventos importantes, errores y excepciones
  - Pasos críticos en la ejecución
  - Inicio y fin de operaciones complejas o largas
- Los logs son útiles para desarrolladores, administradores y usuarios
- Comentario al margen: ¡Los logs no reemplazan al testing!
- Mucho mejor que System.out.println, que es “rapido&sucio”
  - Define jerarquía de “labels”
  - Activar/Desactivar logs (sin tocar código)
  - Generar reportes en varios formatos (txt, json,xml) y destinos (file, screen, socket)

java.util.logging

- La aplicación
  - Configura al framework
  - Manda mensajes a objetos Logger
- El Framework se encarga de:
  - Como se crean, organizan y recuperan esos objetos
  - Como se configuran
  - Como se activan y desactivan
  - A que prestan atención y a que no
  - Cómo se formatean los logs
  - A donde se envían los logs



Loggers:
>ecommers
>ecommers.customers

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

public class SimpleLoggingExample {

    private static final Logger logger = Logger.getLogger(SimpleLoggingExample.class.getName()

    public static void main(String[] args) {
        logger.info("Application started");

        try {
            int result = 10 / 0; // Simulate an error
        } catch (ArithmeticException e) {
            logger.severe("An error occurred: " + e.getMessage());
        }

        logger.info("Application finished");
    }
}
```

```
May 09, 2025 11:07:30 AM SimpleLoggingExample main
INFO: Application started
May 09, 2025 11:07:30 AM SimpleLoggingExample main
SEVERE: An error occurred: / by zero
May 09, 2025 11:07:30 AM SimpleLoggingExample main
INFO: Application finished
```

Variante de uso de Loggers

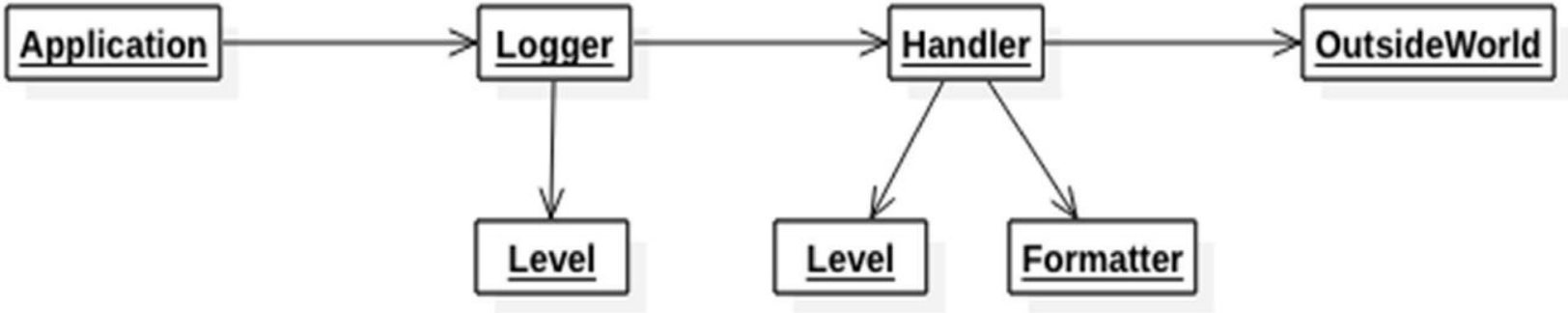
```java
public class Sandbox {
    public static void main(String[] args) throws IOException {
        Logger.getLogger("app.main").addHandler(new FileHandler("log.txt"));
        Logger.getLogger("app.main").log(Level.INFO, "App iniciada");
        try {
            // Acá que hace algo que "podría" resultar en una excepción
            int explodesForSure = 1 / 0;
        } catch (Exception ex) {
            Logger.getLogger("app.main").log(Level.SEVERE, "Explotó!", ex);
        }
        Logger.getLogger("app.main").log(Level.INFO, "App terminada");
    }
}
```

Logger

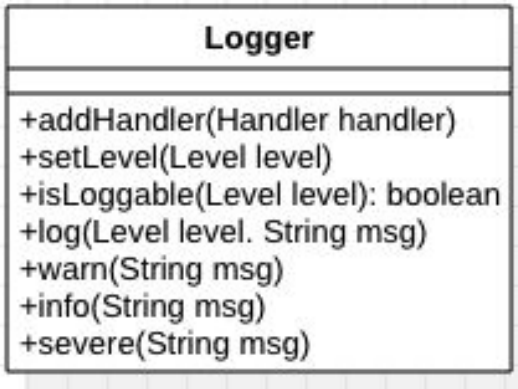
- mantiene un “registry” de sus instancias.
- getLogger() es un lazy-initializer

Arquitectura visible

- Logger: objeto al que le pedímos que emita un mensaje de log
- Handler: encargado de enviar el mensaje a donde corresponda
- Level: indica la importancia de un mensaje y es lo que mira un
- Logger y un Handler para ver si le interesa
- Formater: determina cómo se ”presentará” el mensaje



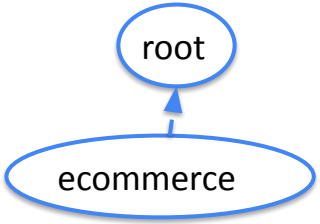
Logger

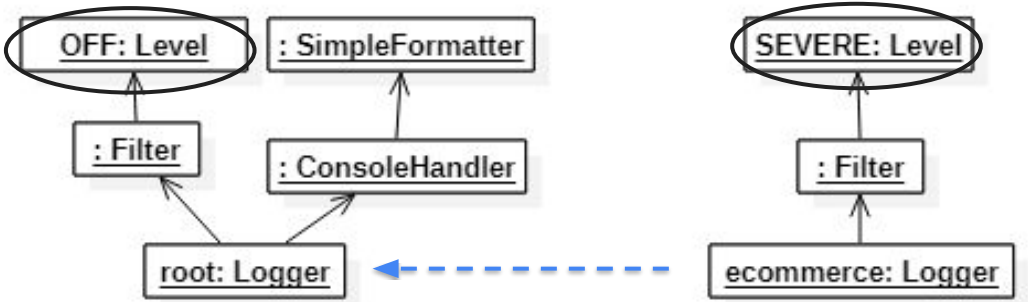


- Podemos definir tantos como necesitemos
  - Instancias de la clase Logger
  - Las obtengo con Logger.getLogger(String nombre)
- Cada uno con su filtro y handler/s
- Se organizan en un árbol (en base a sus nombres)
  - Heredan configuración de su padre (handlers y filters)
- log(Level, String) agrega un mensaje al log
  - Alternativamente uso warn(), info(), severe() …

Ejemplo avanzado

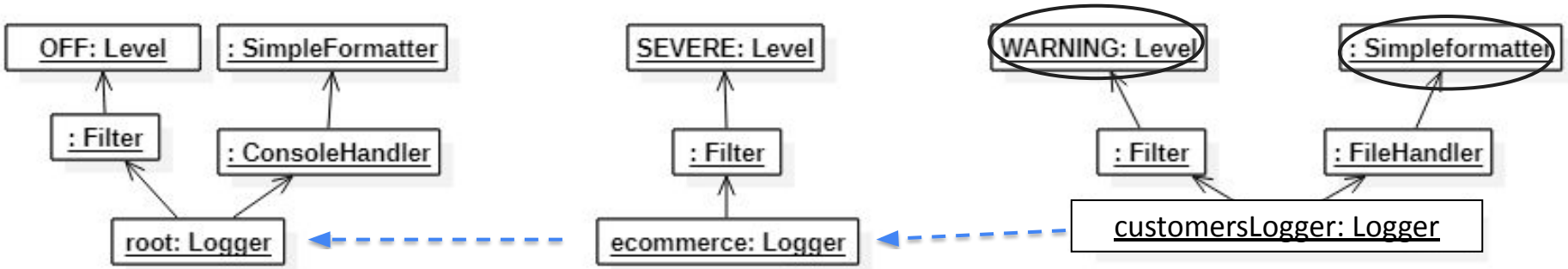
```
//Loggers apagados por defecto
Logger.getLogger("").setLevel(Level.OFF);

//Loggers encendidos en nivel SEVERE para ecommerce
//Utilizará un ConsoleHandler y un SimpleFormatter
Logger  ecommerce = Logger.getLogger("ecommerce");
ecommerce .setLevel(Level.SEVERE);
```





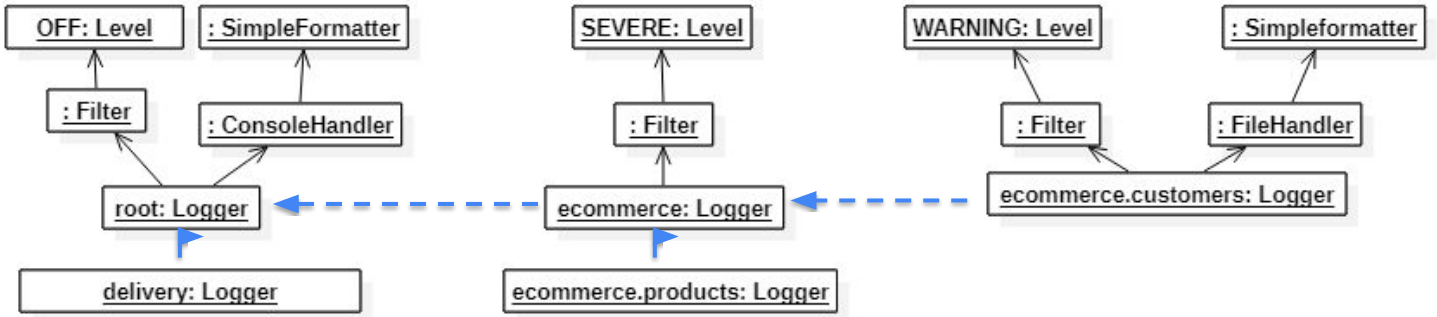
Ejemplo avanzado

```
//Loggers apagados por defecto
```




```
//Logger del servicio de customers de ecommerce, encendido en nivel WARNING
//con destino un archivo, en formato simple texto
Logger customersLogger = Logger.getLogger("ecommerce.customers");
customersLogger.setLevel(Level.WARNING);
FileHandler customersLoggerHandler = new FileHandler("ecommerce-customers.log");
customersLoggerHandler.setFormatter(new SimpleFormatter());
customersLogger.addHandler(customersLoggerHandler);
```

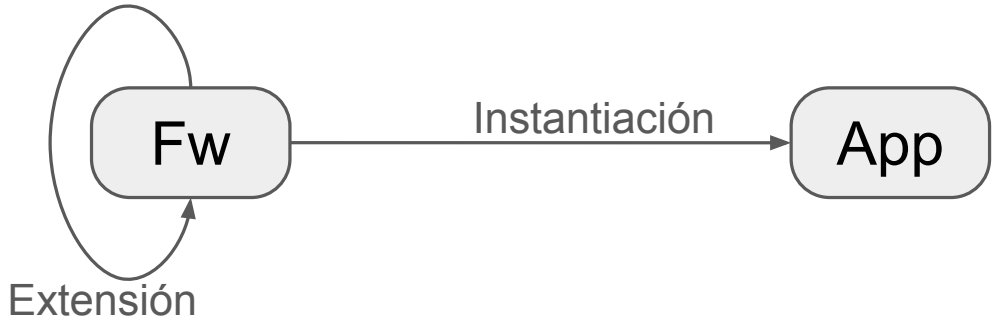
Ejemplo avanzado




```
// Este logger hereda del raíz y no define nada propio, por lo tanto ignora el warning
Logger.getLogger("delivery").log(Level.WARNING, "Error in delivery");

// Este logger hereda de ecommerce por lo tanto ignora el warning
Logger.getLogger("ecommerce.products").log(Level.WARNING, "Stock inconsistency detected");

// A este logger le interesa el warning, que termina en un archivo con formato simple
Logger.getLogger("ecommerce.customers").log(Level.WARNING, "Stock inconsistency detected");
```

Extendiendo el framework

- Y, mirando adentro, puedo agregar nuevas clases de Formater, Handler y Filter
  - Nuevo Formatter: Subclasifico la clase abstracta Formatter o alguna de sus subclases
  - Nuevo Handler: Subclasifico la clase abstracta Handler o alguna de sus subclases
  - Nuevo Filter: Implemento la interfaz Filter
- Esto no es hacking, sino algo previsto por los diseñadores

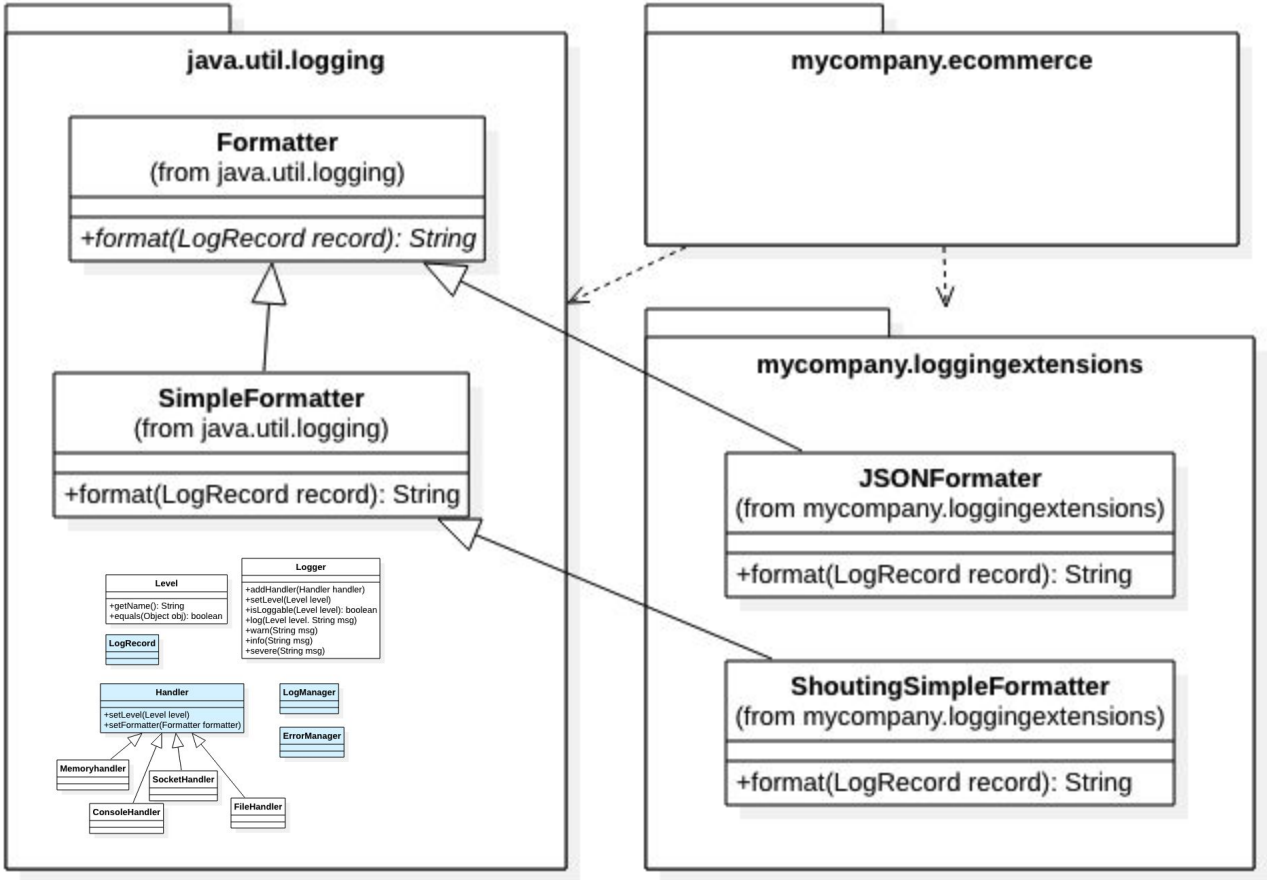


Nuevos Formatters

```java
public class ShoutingSimpleFormatter extends SimpleFormatter {
    @Override
    public String format(LogRecord record) {
        // SHOUTING WITH ALL UPPERCASE
        return super.format(record).toUpperCase();
    }
}
```

```java
public class JSONFormatter extends Formatter {
    @Override
    public String format(LogRecord record) {
        // Do whatever necesssary to represent record as
        // a JSON formatted string and return it
        return "...";
    }
}
```

Desarrolladores
del framework



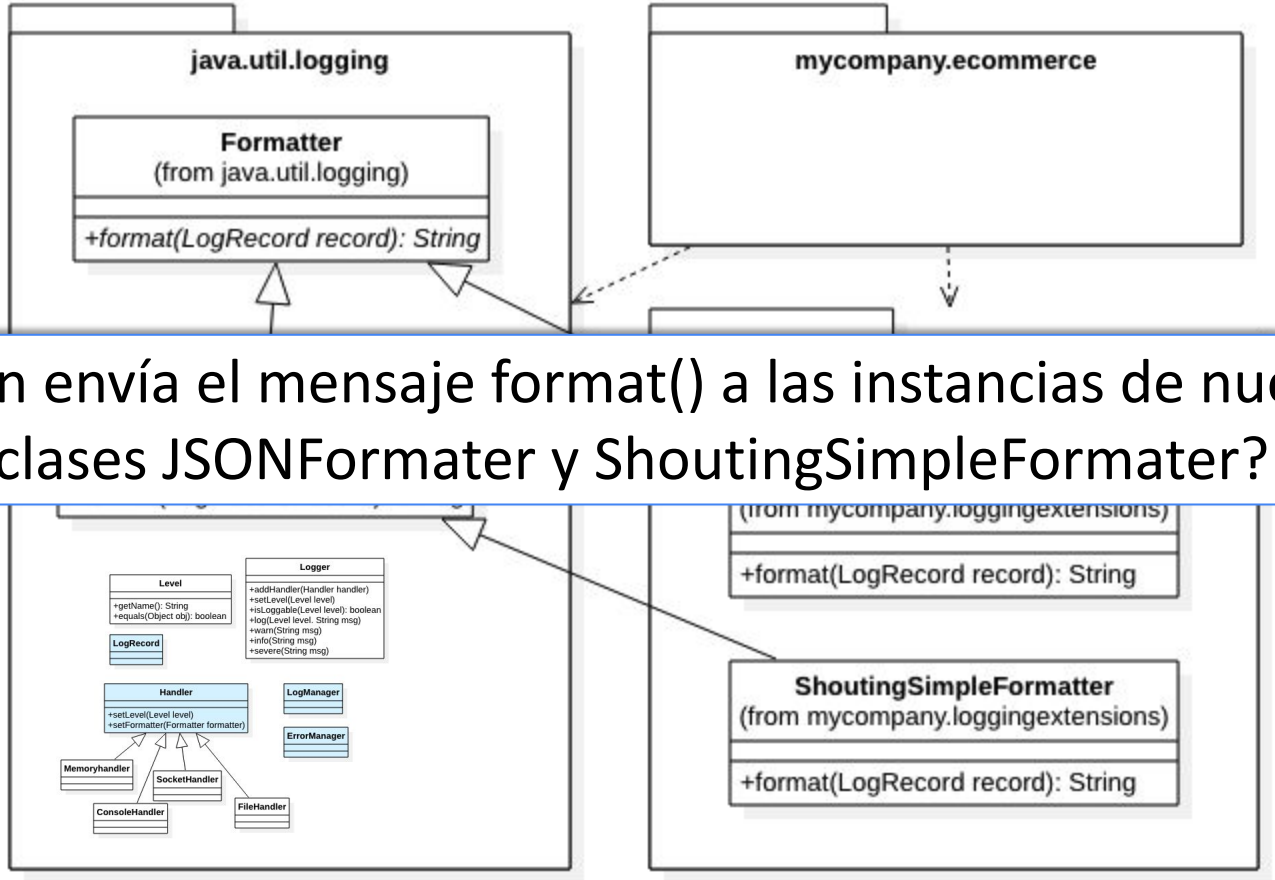
Desarrolladores
de aplicaciones

Mejoradores
del
framework

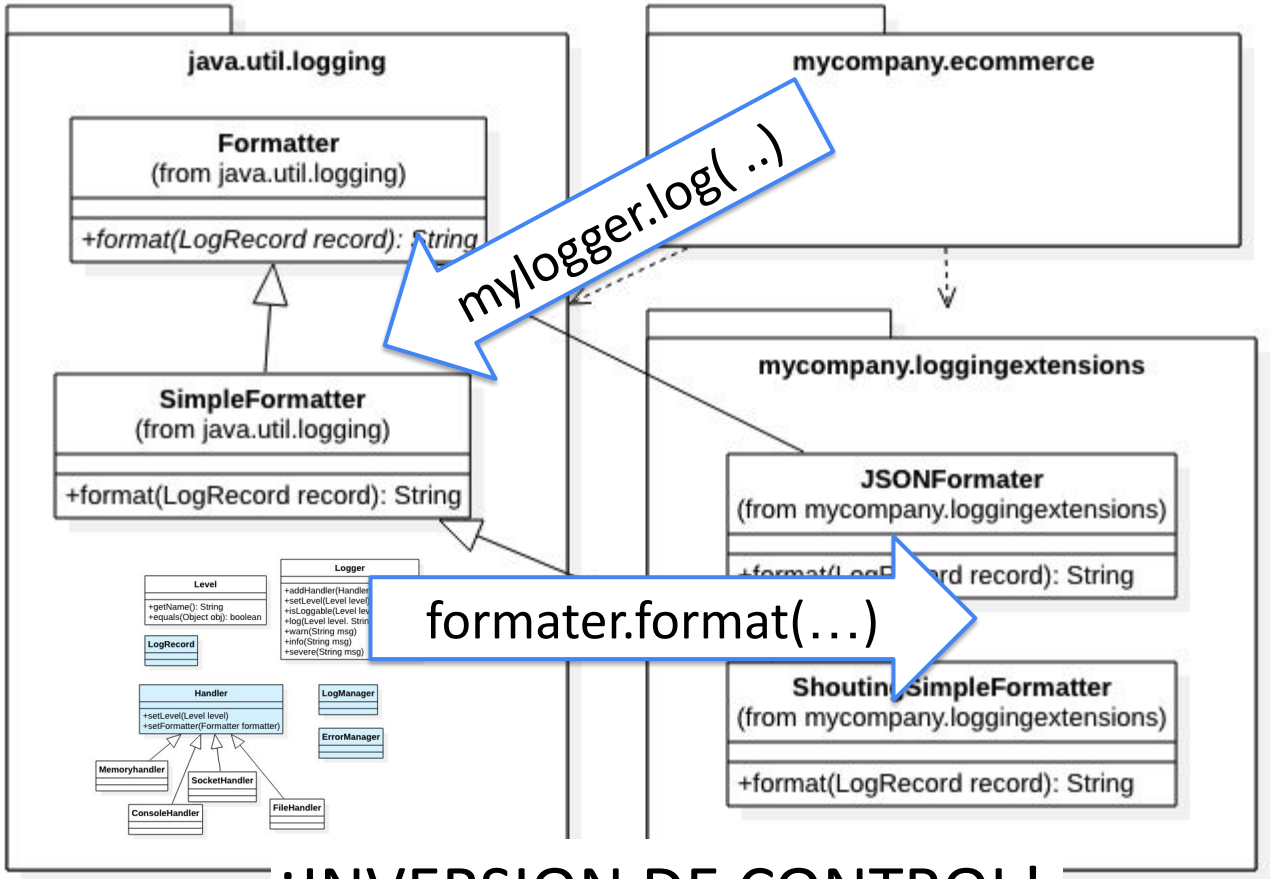
Desarrolladores del framework

Desarrolladores de aplicaciones

Mejoradores del framework



¿Quien envía el mensaje format() a las instancias de nuestras clases JSONFormater y ShoutingSimpleFormater?


Desarrolladores del framework
java.util.logging
Formatter
(from java.util.logging)
+format(LogRecord record): String
SimpleFormatter
(from java.util.logging)
+format(LogRecord record): String
mylogger.log(..)
mycompany.ecommerce
mycompany.loggingextensions
JSONFormater
(from mycompany.loggingextensions)
formater.format(…)
(from mycompany.loggingextensions)
+format(LogRecord record): String
Desarrolladores de aplicaciones
Mejoradores del framework
¡INVERSION DE CONTROL!

Resumiendo

- Frameworks presentan una manera de reuso que supera el reuso (solo de código)
- Un Framework dicta la manera de ejecutar un programa (execution thread)
- El diseño de un framework foco en una manera de describir un dominio o en las cosas importantes de un dominio
- Frameworks vistos en OO2
  - SingleThreadTCPServer: Un servidor como extensión de una clase loop + sesión (singular)
  - Tcp.server.reply: loop(sesión(msgHandler). Sesión singular o multiple. Diferentes MsgHandler
  - Java.util.logging: jerarquía de Labels + composición de filtros, formatos y salidas
- Diseños de Frameworks
  - FrozenSpots: partes del diseño que no cambian
  - HotSpots: elementos del diseño pensadas para adaptarse
    - Hook Methods
    - Hook Classes (hoy en día es poco usual que existan)
  - El diseño se va adaptando a los problemas comunes en un dominio
    - Problemas comunes ⇒ design patterns
      - En CajaBlanca. Loop de control suele ser Template Method.
      - En tcp.server.reply MessageHandler puede ser Strategy o Command (GoF)
    - Si en el diseño de un framework identificamos un patrón, la estructura correspondiente seguramente es un hotspot.

Resumiendo

- Tipos de Instanciación
  - Caja Blanca: las instanciaciones modifican o extienden el códido fuente (loop de control + hook clases)
  - Caja Negra: las instanciaciones se basan en configuraciones

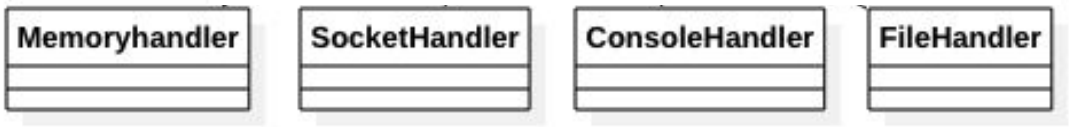
Material en Moodle

- Lectura sobre Frameworks: https://catedras.linti.unlp.edu.ar/mod/imscp/view.php?id=41942
- Código fuente de tcp.server.reply
  - Framework
  - Ejemplos de instanciación
- Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks - D. Roberts, R. Johnson
- Hot-Spot-Driven Framework Development - W. Pree

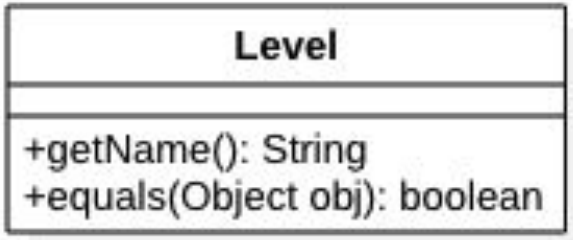
Material adicional java.util.logging

Handler

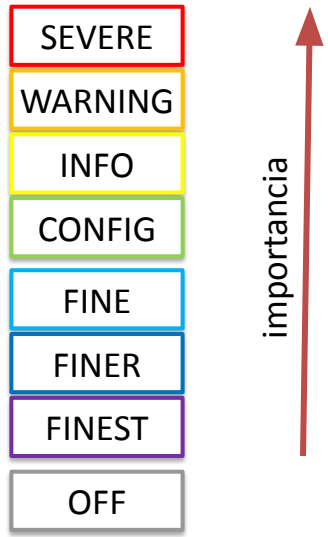
- Recibe los mensajes del Logger y determina como “exportarlos”
- Instancias de MemoryHandler, ConsoleHandler, FileHandler, o SocketHandler
- Puede filtrar por nivel
- Tiene un Formatter



Level



- Representa la importancia de un mensaje
- Cada vez que pido que se loggee algo, debo indicar un nivel
- Los Loggers y Handler comparan el nivel de un cada mensaje con el suyo para decidir si les interesa o no
- Si te interesa un nivel, también te interesan los que son más importantes que ese
- Hay niveles predefinidos, en variables estáticas de la clase Level (p.e., Level.OFF)



Formatter

- El Formatter recibe un mesaje de log (un objeto) y lo transforma a texto
- Son instancias de: SimpleFormatter o XMLFormatter
- Cada handler tiene su formatter
  - Los FileHandler tienen un XMLFormatter por defecto
  - Los ConsoleHandler tienen un SimpleFormatter por defecto