

Larson



"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 instanciaación
- 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

```
1 import java.io.PrintWriter;
2
3 public class EchoServer extends SingleThreadTCPServer {
4
5     public void handleMessage(String message, PrintWriter out) {
6         out.println(message);
7     }
8
9     Run | Debug
10    public static void main(String[] args) {
11        new EchoServer().startLoop(args);
12    }
13
14 }
```

SingleThreadTCPServer  
(hotspot herencia)

```
1 import tcp.server.reply.*;
2
3 public class EchoApp {
4
5     Run | Debug
6     public static void main(String[] args) {
7
8         new TCPControlLoop(new SingleConnectionHandler(new EchoHandler())).startLoop(args);
9
10    }
```

tcp.server.reply  
(hotspot composición)

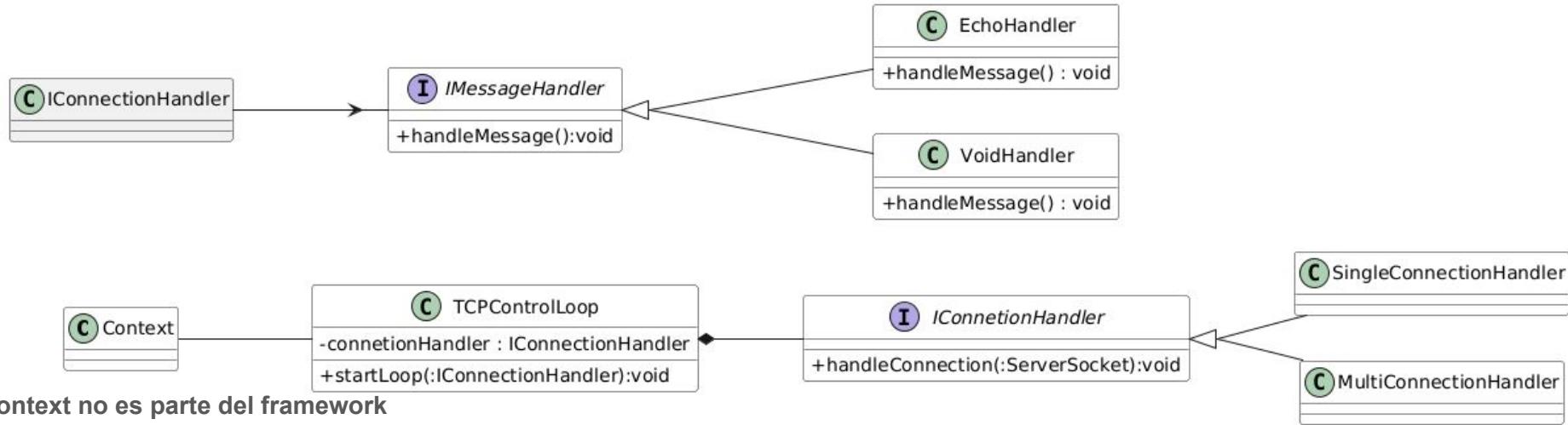
```
1 import tcp.server.reply.*;  
2  
3 public class MultiEchoApp {  
4     Run | Debug  
5     public static void main(String[] args) {  
6         new TCPControlLoop(new MultiConnectionHandler(new EchoHandler())).startLoop(args);  
7     }  
8 }  
9  
10
```

tcp.server.reply  
multisession, echo handler

```
1 import tcp.server.reply.*;  
2  
3 public class TestApp {  
4     Run | Debug  
5     public static void main(String[] args) {  
6         new TCPControlLoop().startLoop(args);  
7     }  
8 }  
9  
10 }
```

tcp.server.reply  
single session, void handler

# tcp.server.reply



Context no es parte del framework

## FrozenSpot:

Condición de corte de la conexión  
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)

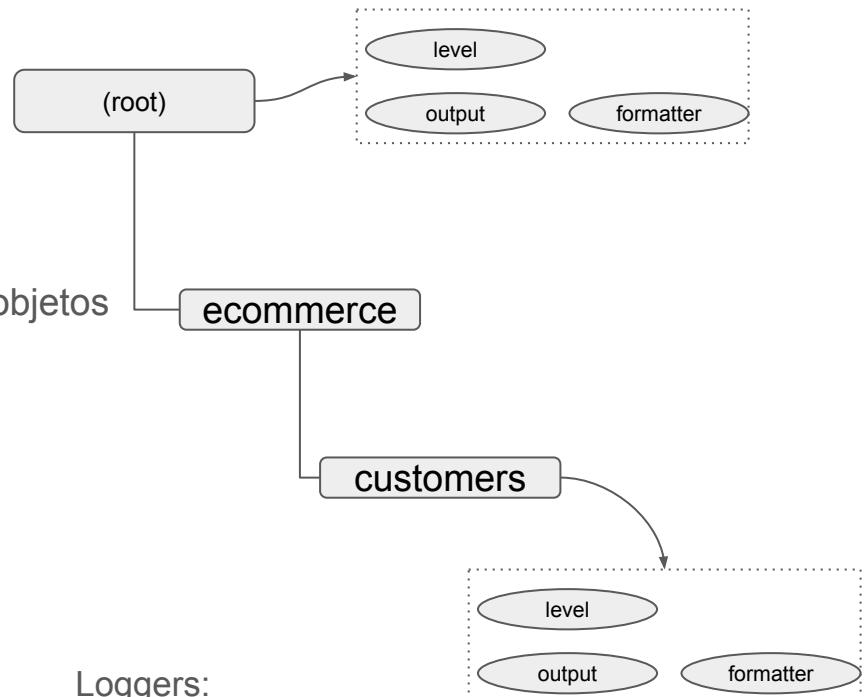
```
code/phoenix-chat-example $ mix phx.server
[info] Running ChatWeb.Endpoint with Cowboy using http://0.0.0.0:4000
09:50:37 - info: compiled 6 files into 2 files, copied 3 in 1.7 sec
[warn] Ignoring unmatched topic "topic:subtopic" in ChatWeb.UserSocket
[info] JOIN "chat_room:lobby" to ChatWeb.ChatRoomChannel
  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
```

# 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:  
>*ecommerce*  
>*ecommerce.customers*

```
1 import java.util.logging.Logger;
2
3
4 public class SimpleLoggingExample {
5
6     private static final Logger logger = Logger.getLogger(SimpleLoggingExample.class.getName());
7
8     public static void main(String[] args) {
9         logger.info("Application started");
10
11         try {
12             int result = 10 / 0; // Simulate an error
13         } catch (ArithmaticException e) {
14             logger.severe("An error occurred: " + e.getMessage());
15         }
16
17         logger.info("Application finished");
18     }
19 }
```

```
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

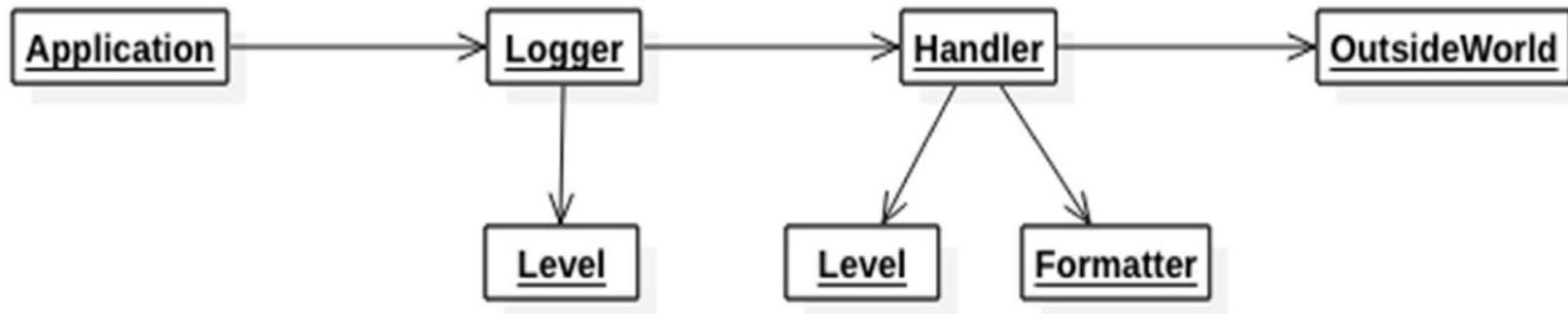
```
public class Sandbox {  
    public static void main(String[] args) throws IOException {  
        Logger.getLogger("app.main").addHandler(new FileHandler("log.txt"));  
        Logger.getLogger("app.main").log(Level.INFO, "App iniciada");  
        try {  
            // Acá que hace algo que "podría" 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
- Formatter: 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 parente (handlers y filters)
- log(Level, String) agrega un mensaje al log
  - Alternativamente uso warn(), info(), severe() ...

Logger
+addHandler(Handler handler)
+setLevel(Level level)
+isLoggable(Level level): boolean
+log(Level level, String msg)
+warn(String msg)
+info(String msg)
+severe(String msg)

# Ejemplo avanzado

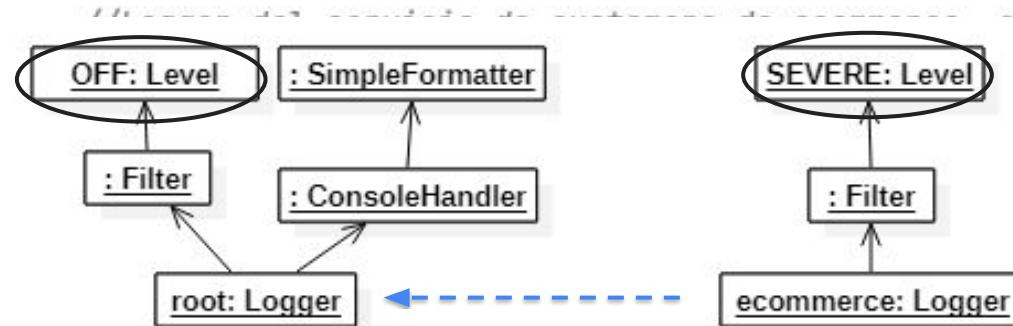
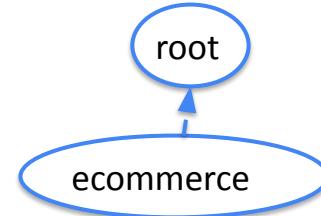
```
//Loggers apagados por defecto
```

```
Logger.getLogger("").setLevel(Level.OFF);
```

```
//Loggers encendidos en nivel SEVERE para ecommerce
```

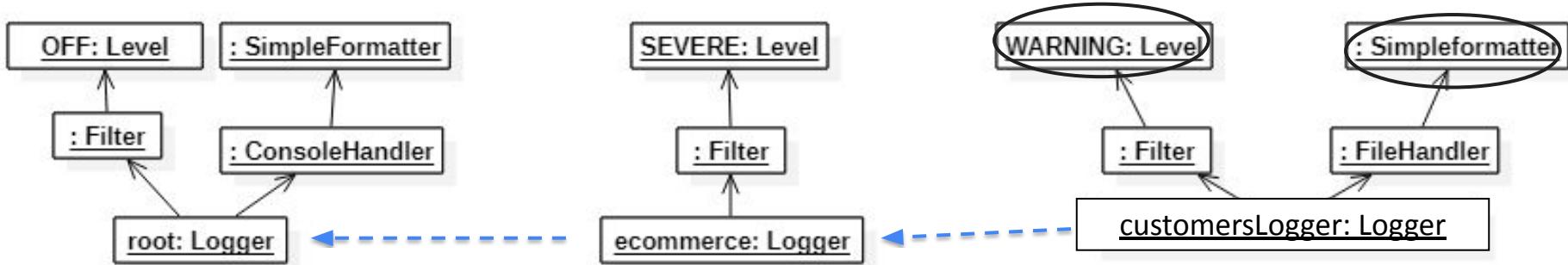
```
//Utilizará un ConsoleHandler y un SimpleFormatter
```

```
Logger ecommerce = Logger.getLogger("ecommerce");
ecommerce.setLevel(Level.SEVERE);
```



# Ejemplo avanzado

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

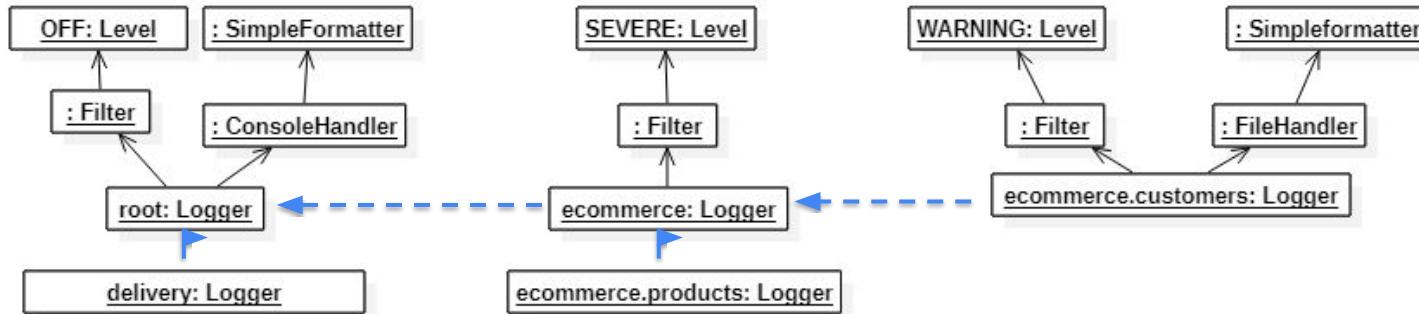


```
//Logger del servicio de customers de ecommerce, encendido en nivel WARNING
```

```
//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 subclasses
  - Nuevo Handler: Subclasifico la clase abstracta Handler o alguna de sus subclasses
  - Nuevo Filter: Implemento la interfaz Filter
- Esto no es hacking, sino algo previsto por los diseñadores

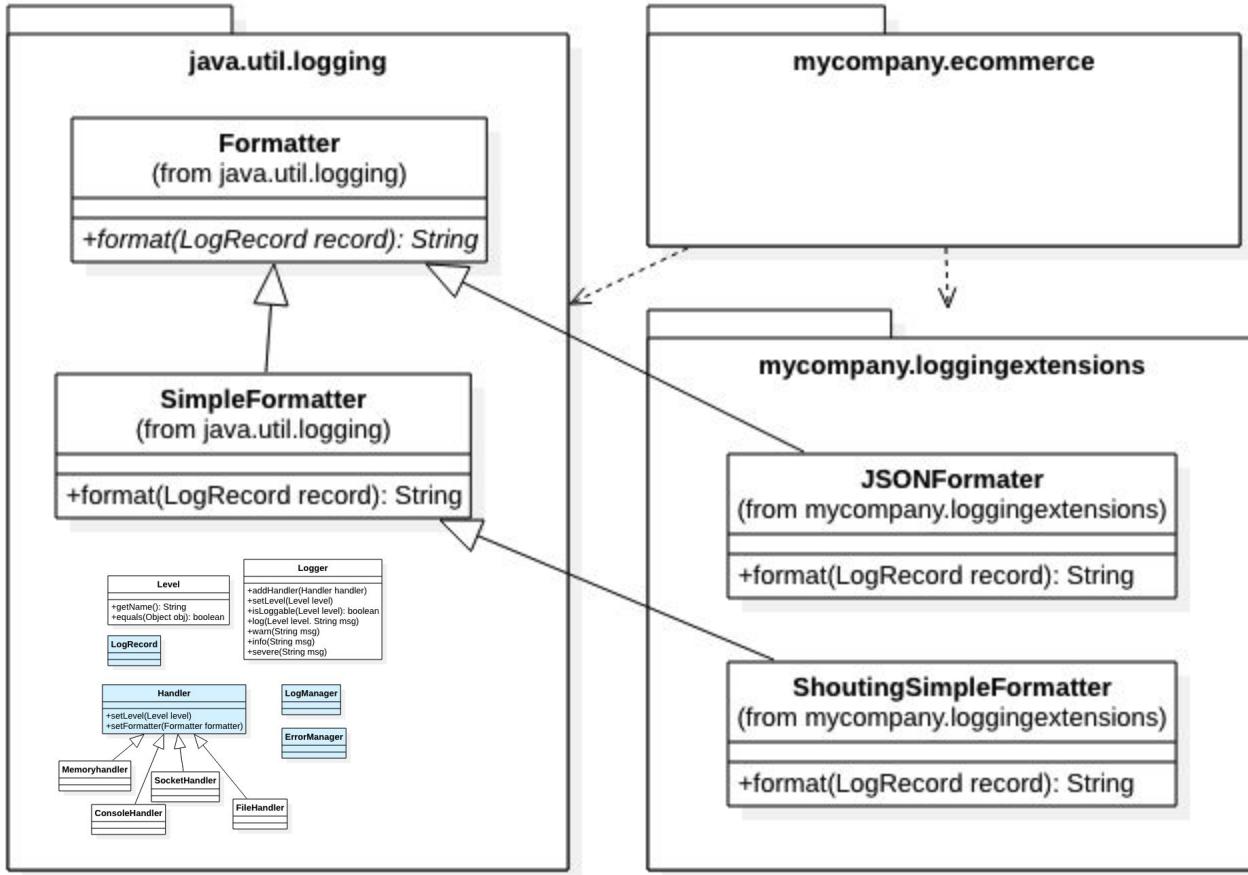
# Nuevos Formatters

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

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



Desarrolladores  
del framework



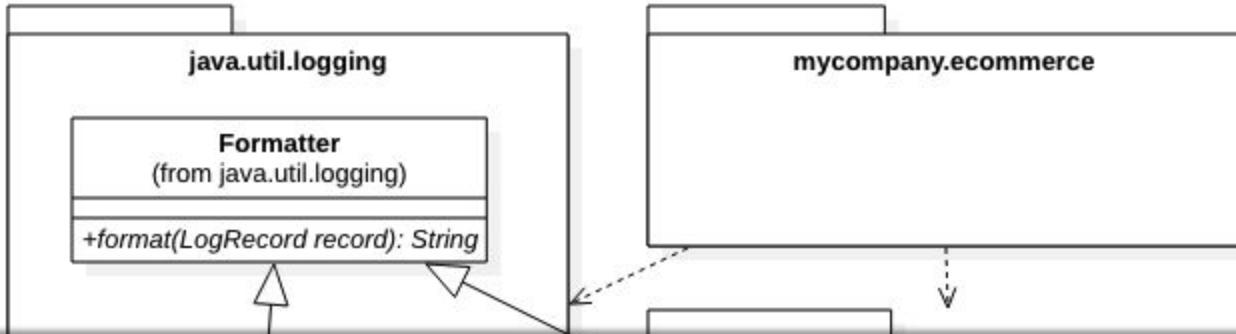
Desarrolladores  
de aplicaciones



Mejoradores  
del  
framework

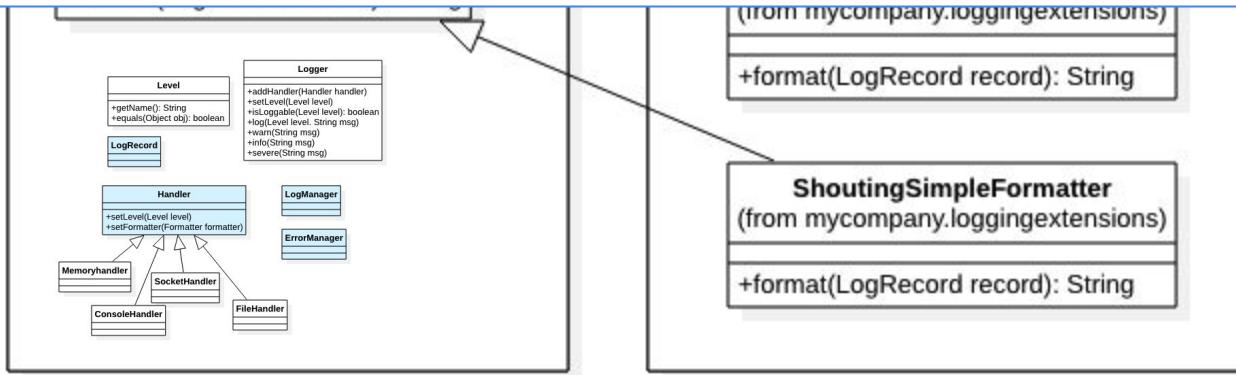


Desarrolladores  
del framework



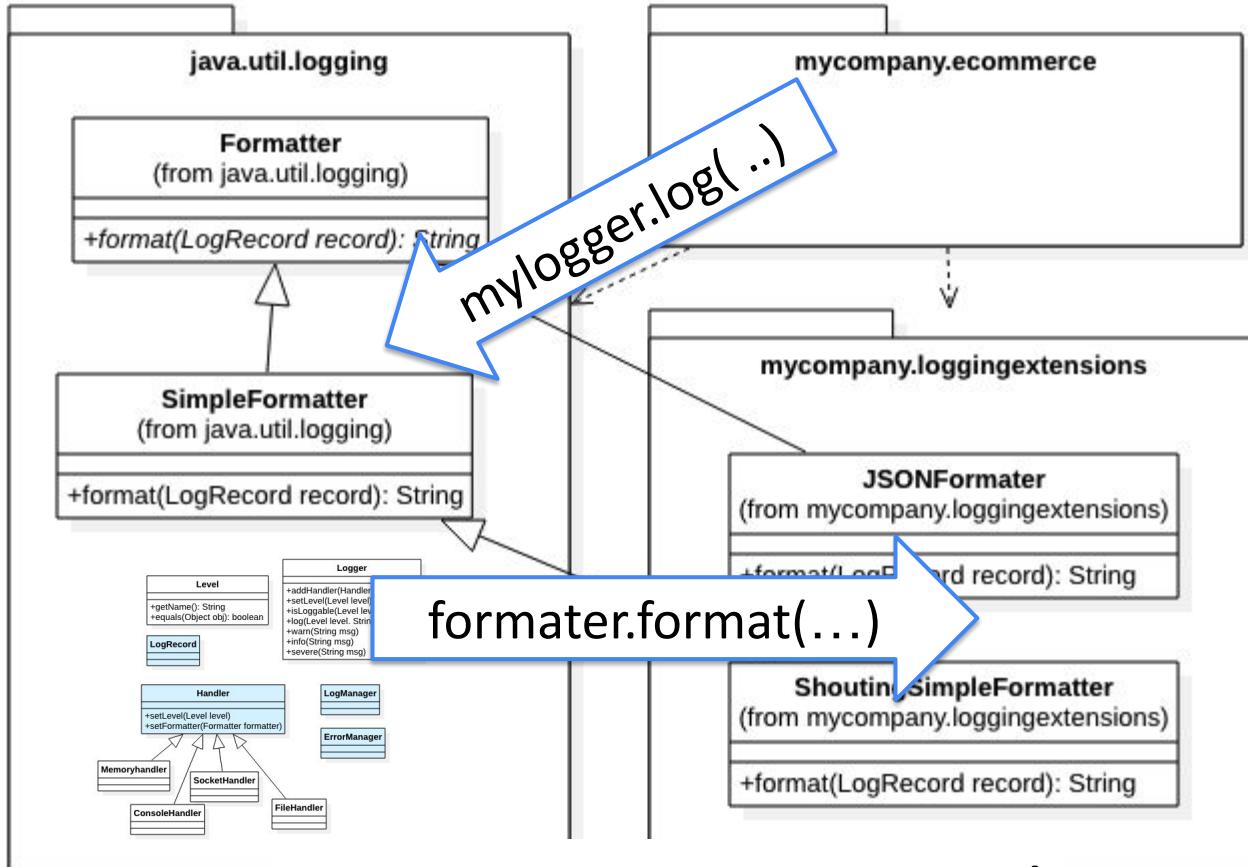
Desarrolladores  
de aplicaciones

¿Quién envía el mensaje `format()` a las instancias de nuestras clases `JSONFormatter` y `ShoutingSimpleFormatter`?



Mejoradores  
del  
framework

  
Desarrolladores  
del framework



  
Desarrolladores  
de aplicaciones

  
Mejoradores  
del  
framework

**¡INVERSIÓN DE CONTROL!**

# Resumiendo

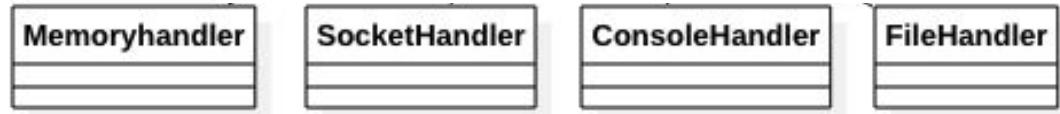
- Frameworks presentan una manera superior de reuso (la manera de ejecutar un programa)
  - SingleThreadTCPServer: Un servidor como extensión de una clase loop + sesión (singular)
  - Tcp.server.reply: loop(sesión(msgHandler)). Sesión singular o multiple. Diferentes MsgHandler
  - Java.util.logging: jerarquía de Labels + composición de filtros, formatos y salidas
- Tipos de Instanciación
  - Caja Blanca: las instanciaciones modifican o extienden el código fuente (loop de control + hook clases)
  - Caja Negra: las instanciaciones se basan en configuraciones
- 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)
- Material en Moodle
- <https://catedras.linti.unlp.edu.ar/mod/imscp/view.php?id=39269>
- 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

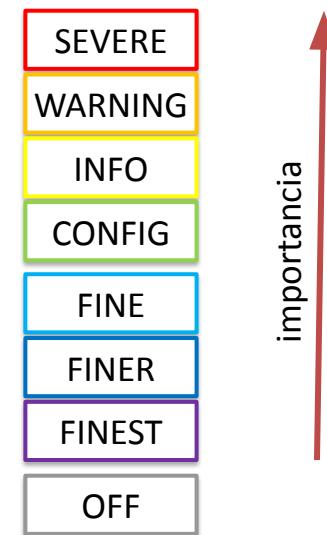
- Recibe los mensajes del Logger y determina como “exportarlos”
- Instancias de MemoryHandler, ConsoleHandler, FileHandler, o SocketHandler
- Puede filtrar por nivel
- Tiene un Formatter



# Level

Level
+getName(): String +equals(Object obj): boolean

- 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 mensaje 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

