

Reúso de software (repaso)



Reúso

- ¿qué es lo que reusamos?
- ¿por qué el reúso es un tema importante en Ing. De Software?
- ¿qué dificultades encontramos?
- ¿qué alternativas de reúso tenemos?

¿qué reusamos?

- Conceptos e ideas que funcionan
 - P.e., Compartir en las redes sociales; ayuda de contexto...
- Diseños o estrategias de diseño (que imitamos, implementamos)
 - P.e., Patrones de diseño, algoritmos de búsqueda, arquitecturas...
- Funciones y estructuras de datos
 - P.e., de Colecciones, de Fechas, de Archivos,
- Componentes y Servicios (que invocamos)
 - P.e., Twitter API, OpenAI API, Google Maps, ...
- Aplicaciones completas que adaptamos e integramos
 - P.e., Drupal/Wordpress; Minecraft, ...

¿qué reusamos?

- Conceptos e ideas que funcionan
 - P.e., Compartir en las redes sociales; ayuda de contexto...
- Diseños o estrategias de diseño (que imitamos, implementamos)
 - P.e., Patrones de diseño, algoritmos de búsqueda, arquitecturas...
- Funciones y estructuras de datos
 - P.e., de Colecciones, de Fechas, de Archivos,
- Componentes y Servicios (que invocamos)
 - P.e., Twitter API, OpenAI API, Google Maps, ...
- Aplicaciones completas que adaptamos e integramos
 - P.e., Drupal/Wordpress; Minecraft, ...

¿qué reusamos?

- Conceptos e ideas que funcionan
 - P.e., Compartir en las redes sociales; ayuda de contexto...
- Diseños o estrategias de diseño (que imitamos, implementamos)
 - P.e., Patrones de diseño, algoritmos de búsqueda, arquitecturas...
- **Funciones y estructuras de datos**
 - P.e., de Colecciones, de Fechas, de Archivos,
- Componentes y Servicios (que invocamos)
 - P.e., Twitter API, OpenAI API, Google Maps, ...
- Aplicaciones completas que adaptamos e integramos
 - P.e., Drupal/Wordpress; Minecraft, ...

¿qué reusamos?

- Conceptos e ideas que funcionan
 - P.e., Compartir en las redes sociales; ayuda de contexto...
- Diseños o estrategias de diseño (que imitamos, implementamos)
 - P.e., Patrones de diseño, algoritmos de búsqueda, arquitecturas...
- Funciones y estructuras de datos
 - P.e., de Colecciones, de Fechas, de Archivos,
- **Componentes y Servicios (que invocamos)**
 - P.e., Twitter API, OpenAI API, Google Maps, ...
- Aplicaciones completas que adaptamos e integramos
 - P.e., Drupal/Wordpress; Minecraft, ...

¿qué reusamos?

- Conceptos e ideas que funcionan
 - P.e., Compartir en las redes sociales; ayuda de contexto...
- Diseños o estrategias de diseño (que imitamos, implementamos)
 - P.e., Patrones de diseño, algoritmos de búsqueda, arquitecturas...
- Funciones y estructuras de datos
 - P.e., de Colecciones, de Fechas, de Archivos,
- Componentes y Servicios (que invocamos)
 - P.e., Twitter API, OpenAI API, Google Maps, ...
- Aplicaciones completas que adaptamos e integramos
 - P.e., Drupal/Wordpress; Minecraft, ...

¿por qué?



- Para aumentar la productividad
 - Reducir los costos de desarrollo y mantenimiento
 - Reducir los tiempos de entrega y puesta en el mercado
 - Agilidad de mercado y customización en masa



- Para aumentar la calidad
 - El reúso aprovecha mejoras y correcciones – el tiempo da madurez y confiabilidad
 - Se reaprovecha el conocimiento de los especialistas (que queda encapsulado en componentes reusables)
 - Ayuda a la implementación e imposición de estándares

¿por qué?



- Para aumentar la productividad
 - Reducir los costos de desarrollo y mantenimiento
 - Menores tiempos de entrega y puesta en el mercado
 - Agilidad de mercado y customización en masa



- Para aumentar la calidad
 - El reúso aprovecha mejoras y correcciones – el tiempo da madurez y confiabilidad
 - Se reaprovecha el conocimiento de los especialistas (que queda encapsulado en componentes reusables)

¿qué dificultades tenemos?

- Problemas de mantenimiento si no tenemos control de los componente que reusamos
- Algunos programadores prefieren hacer todo ellos mismos (confianza en lo propio, búsqueda de desafíos)
- Hacer software reusable es mas difícil y costoso (paga a la larga y requiere procesos especiales)
- Encontrar, aprender a usar y adaptar componentes reusables requiere esfuerzo adicional

¿qué dificultades tenemos?

- Dificultades de mantenimiento si no tenemos control de los componente que reusamos
- Algunos programadores prefieren hacer todo ellos mismos (confianza en lo propio, búsqueda de desafíos)
- Hacer software reusable es mas difícil y costoso (paga a la larga y requiere procesos especiales)
- Encontrar, aprender a usar y adaptar componentes reusables requiere esfuerzo adicional

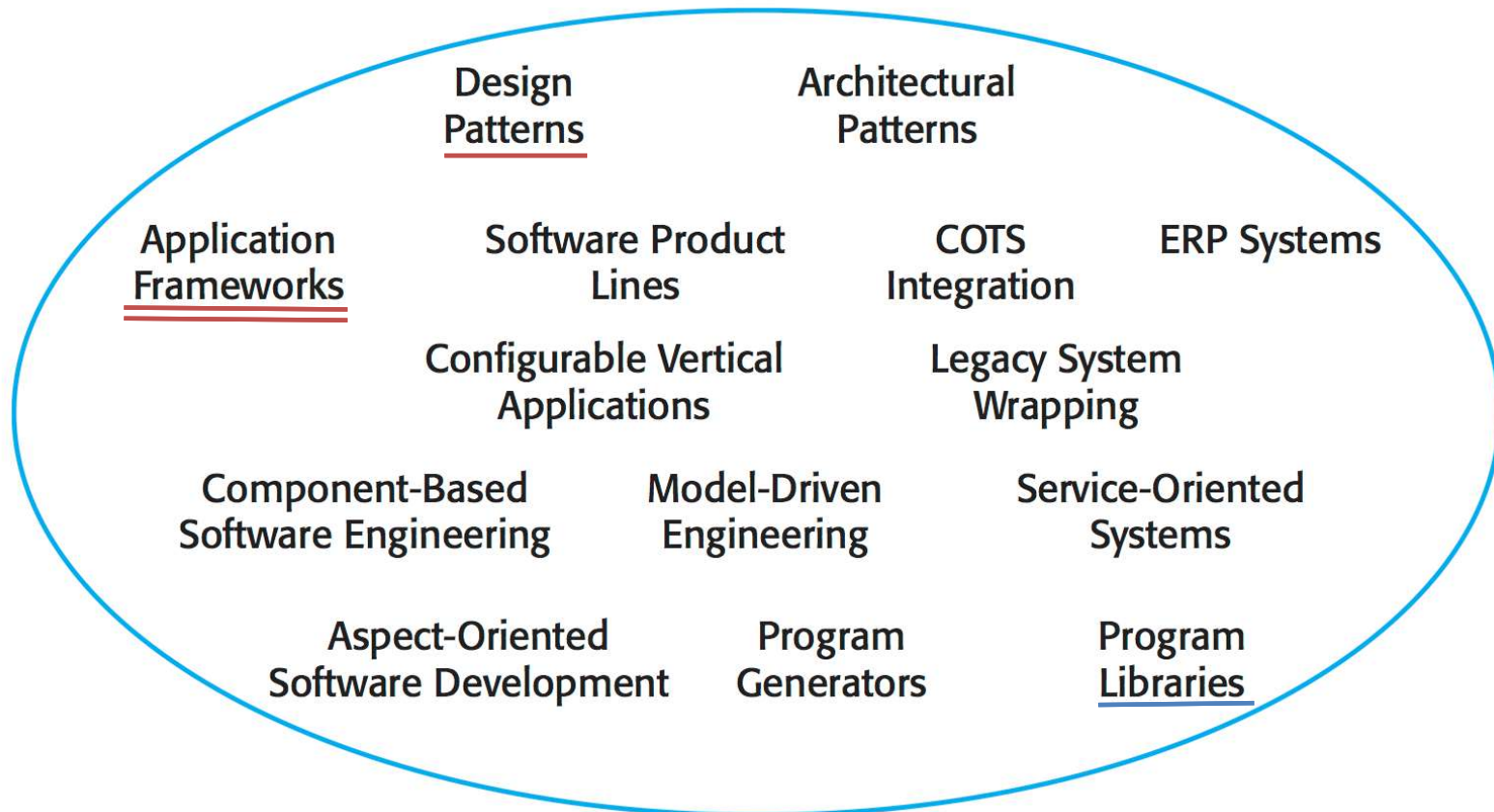
¿qué dificultades tenemos?

- Dificultades de mantenimiento si no tenemos control de los componente que reusamos
- Algunos programadores prefieren hacer todo ellos mismos (confianza en lo propio, búsqueda de desafíos)
- Hacer software reusable es mas difícil y costoso (paga a la larga y requiere procesos especiales)
- Encontrar, aprender a usar y adaptar componentes reusables requiere esfuerzo adicional

¿qué dificultades tenemos?

- Dificultades de mantenimiento si no tenemos control de los componente que reusamos
- Algunos programadores prefieren hacer todo ellos mismos (confianza en lo propio, búsqueda de desafíos)
- Hacer software reusable es mas difícil y costoso (paga a la larga y requiere procesos especiales)
- Encontrar, aprender a usar y adaptar componentes reusables requiere esfuerzo adicional

¿que alternativas tenemos?





LIBRERIAS DE CLASES / TOOLKITS

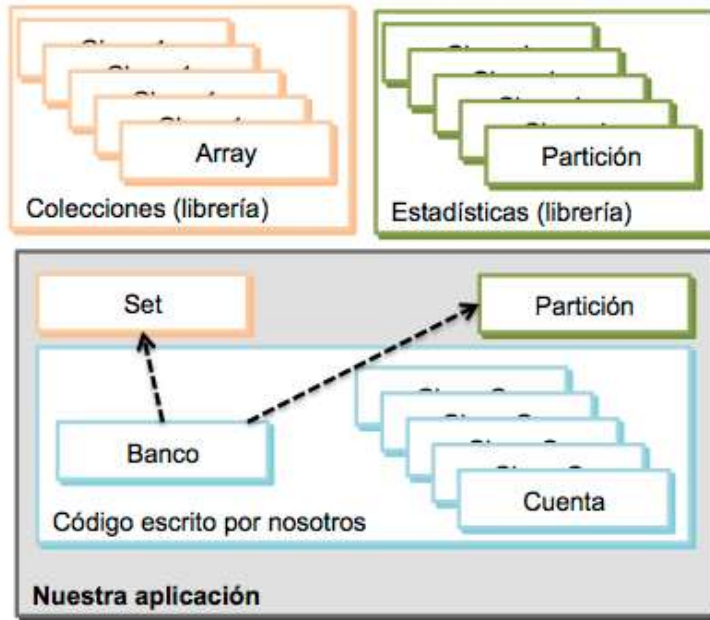
Librería de clases

- Resuelven problemas comunes a la mayoría de las aplicaciones
 - P.e., manejo de archivos, funciones aritméticas, colecciones, fechas,
- Cada clase en la librería resuelve un problema concreto, es independiente del contexto de uso, no espera nada de nuestro código, y generalmente independiente de otras clases en la librería
- Nuestro código controla/usa a los objetos de las librerías

Librería de clases

- Resuelven problemas comunes a la mayoría de las aplicaciones
 - P.e., manejo de archivos, funciones aritméticas, colecciones, fechas,
- Cada clase en la librería resuelve un problema concreto, es independiente del contexto de uso, no espera nada de nuestro código y generalmente independiente de otras clases en la librería
- Nuestro código controla/usa a los objetos de las librerías

Librería de clases



comunes a la mayoría de las
funciones aritméticas,
resuelve un problema
nuevamente del contexto de uso, y
dependiente de otras clases en la

- Nuestro código controla/usa a los objetos de las librerías

Algunas librerías de clases

Java

Codec (apache common)

Clases (hoy, 18) que implementan algoritmos de encoding/decoding algorithms (por ejemplo, phonetic, base64, URL).

Compress (apache common)

Clases para trabajar con archivos tar, zip, bzip2, etc.

Math (apache common)

Clases que implementan componentes autocontenidos de matemáticas y estadísticas.

java.útil.collections

Clases que implementan estructuras de datos (colecciones) y algoritmos para manipularlas



FRAMEWORKS ORIENTADOS A OBJETOS

Frameworks Orientados a Objetos

- Un framework es una *aplicación “semi-completa”, “reusable”, que puede ser especializada para producir aplicaciones a medida...*
- *...un conjunto de clases concretas y abstractas, relacionadas para proveer una arquitectura reusable para una familia de aplicaciones relacionadas...*

Frameworks Orientados a Objetos

- Los desarrolladores incorporan el framework en sus aplicaciones y lo especializan para cubrir sus necesidades ...
- *... el framework define el esqueleto, y el desarrollador define sus propias características para completar el esquelero ...*
- *... (a diferencia de un toolkit o librería) un framework provee una estructura coherente en lugar de un conjunto de “clases útiles” ...*

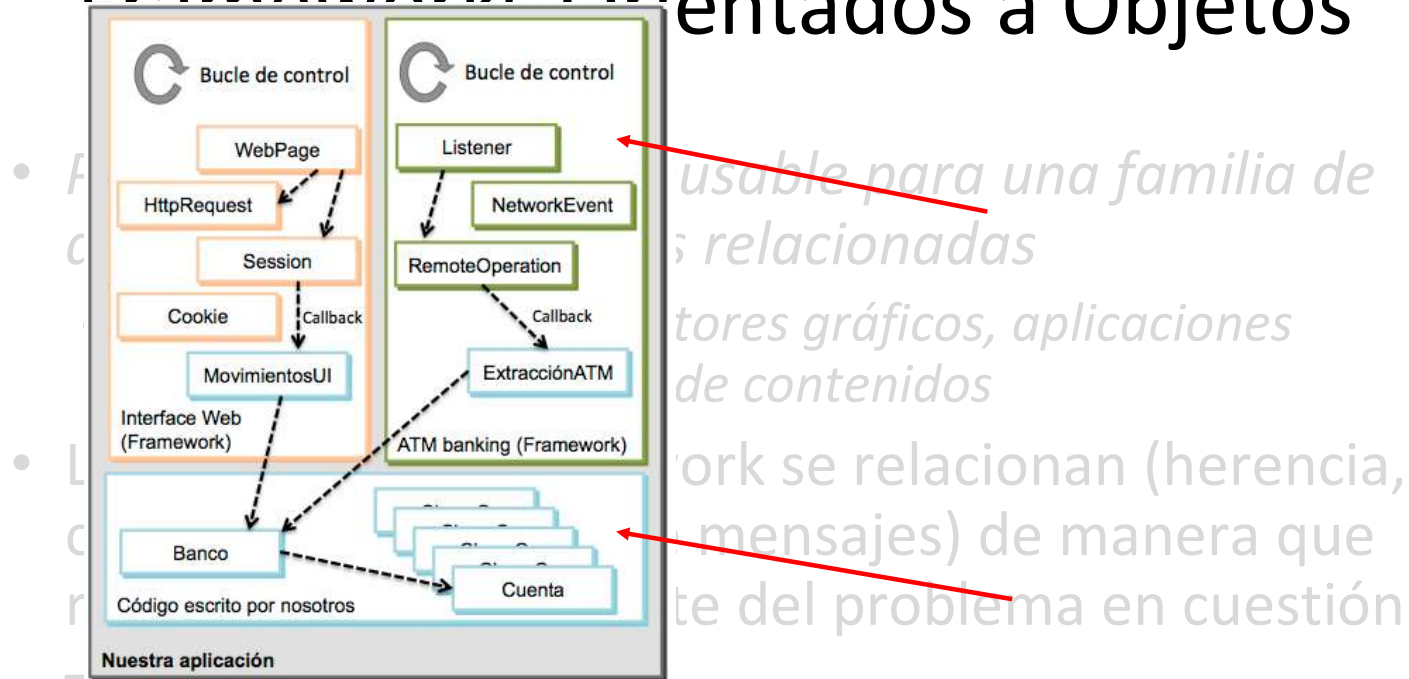
Frameworks Orientados a Objetos

- *Proveer una solución reusable para una familia de aplicaciones/problemas relacionadas*
 - *P.e., interfaces web, editores gráficos, aplicaciones colaborativas, gestores de contenidos*
- Las clases en el framework se relacionan (herencia, conocimiento, envío de mensajes) de manera que resuelven la mayor parte del problema en cuestión – conforman un todo
- El código del framework controla al nuestro

Frameworks Orientados a Objetos

- *Proveer una solución reusable para una familia de aplicaciones/problemas relacionadas*
 - *P.e., interfaces web, editores gráficos, aplicaciones colaborativas, gestores de contenidos*
- Las clases en el framework se relacionan (herencia, conocimiento, envío de mensajes) de manera que resuelven la mayor parte del problema en cuestión
 - conforman un todo
- El código del framework controla al nuestro

Frameworks Orientados a Objetos



- El código del framework controla al nuestro:

INVERSIÓN DE CONTROL

Frameworks de infraestructura

- Estos frameworks ofrecen una infraestructura portable y eficiente sobre la cual construir una gran variedad de aplicaciones
- Algunos de los focos de este tipo de frameworks son: interfaces de usuario (desktop, web, móviles), seguridad, contenedores de aplicación, procesamiento de imágenes, procesamiento de lenguaje, comunicaciones
- Atacan problemas generales del desarrollo de software, que por lo general no son percibidos completamente por el usuario de la aplicación (es decir, resuelven problemas de los programadores, no de los usuarios)
- Es común que se los incluya como parte de plataformas de desarrollo (Java tiene los suyos, al igual de .NET)

Frameworks de integración

- Estos frameworks se utilizan comunmente para integrar componentes de aplicación distribuidos (p.e., la base de datos con la aplicación y ésta con su cliente liviano)
- Los frameworks de integración (o frameworks middleware) estan diseñados para aumentar la capacidad de los desarrolladores de modularizar, reusar y extender su infraestructura de software para que trabaje transparentemente en un ambiente distribuido
- El mercado de los frameworks de integración es muy activo y, al igual que ciertos frameworks de infraestructura, se han vuelto commodities (mercancia de uso común)
- Ejemplos: Object Relational Mapping, Message Oriented Middleware, Orchestration Frameworks

Frameworks de aplicación (o Enterprise)

- Estos frameworks abarcan dominios de aplicación amplios que son pilares fundamentales de las actividades de las empresas
- Ejemplos de frameworks enterprise son aquellos en el dominio de los ERP (Enterprise Resource Planning), CRM (Customer Relationship Management) Gestión de documentos, Cálculos financieros.
- Los problemas que atacan derivan directamente de las necesidades de los usuarios de las aplicaciones y por tanto hacen que el retorno de la inversión en su desarrollo/adquisición sea mas evidente y justificado
- Un framework enterprise puede encapsular el conocimiento y experiencia de muchos años de una empresa, transformandose en la clave de su ventaja competitiva y su mas preciado capital
- Son la base para la implementación de líneas de productos (SPL)

Algunos frameworks conocidos

Java

Hibernate

Framework de integración que resuelve el mapeo de objetos a bases de datos relacionales

jBPM

Framework de aplicación (y suite de herramientas) para construir aplicaciones en las que se modelan, ejecutan, y monitorean procesos de negocios.

jUnit

Framework de aplicación/infraestructura que resuelve la automatización de tests de unidad

libGDX

Framework de aplicación, para construir juegos en Java, multiplataforma.

Spring

Familia de frameworks de infraestructura e integración, enfocando una amplia variedad de necesidades (Testing, Data, Web, etc.)

Frozenspot vs Hotspots

- Frozenspots de un framework
 - Todas las aplicaciones construídas con un mismo framework tienen aspectos en común que no podemos cambiar
- Hotspots de un framework
 - El framework ofrece puntos de extensión que nos permiten introducir variantes y así construir aplicaciones diferentes



Caja blanca vs Caja negra

- Los puntos de extensión pueden implementarse en base a herencia o en base a composición
- A los frameworks que utilizan herencia en sus puntos de extensión, les llamamos de Caja Blanca (Whitebox)
- A los que utilizan composición les llamamos de Caja Negra (blackbox)
- La mayoría de los frameworks están en algún lugar en el medio
 - Algunos usuarios los ven como caja negra
 - Otros como caja blanca

Material adicional

Frameworks: Guía de lectura

TOC

- Frameworks
 - Guía de lectura
 - Preguntas de autoevaluación

<< < ^ > >>

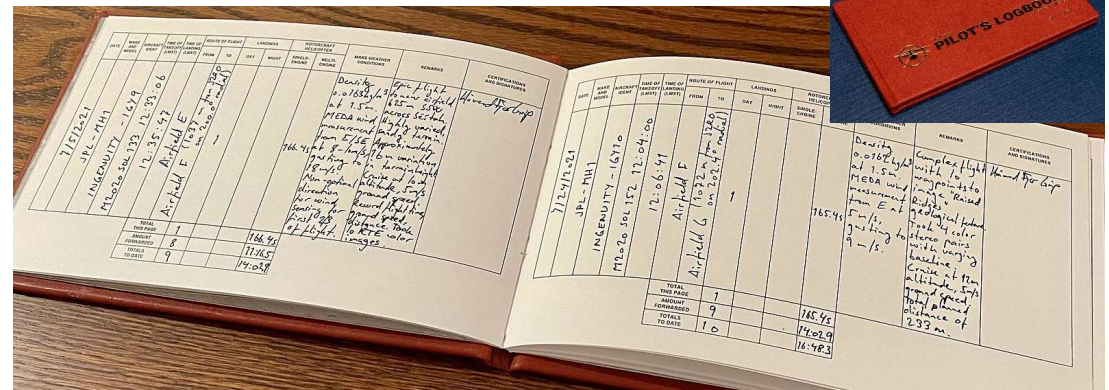
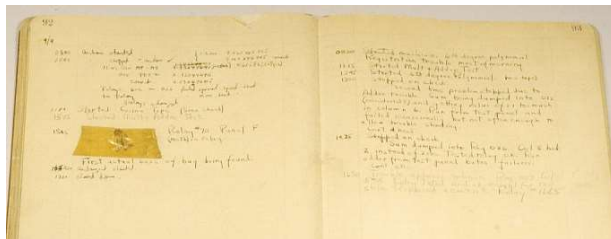
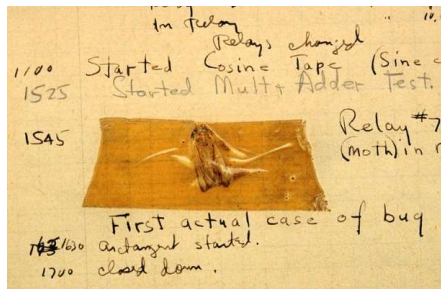
Frameworks

Un framework se puede ver como el resultado de refactorizar una aplicación, no solo para mejorar su calidad interna sino para abstraer lo que es común todas una familia de aplicaciones similares. En ese proceso de refactoring se identifica lo que se mantendrá constante y será, a partir de ese momento, responsabilidad de los desarrolladores del framework y lo que podrá variar de aplicación a aplicación y será responsabilidad de los desarrolladores de aplicaciones que usen el framework.

Para separar la implementación de la parte constante (a lo que llamamos **frozenspot**) y de las partes que varían (a las que llamamos **hotspots**), se introducen en el framework puntos de extensión. Estos tienen la forma de plantillas que ofrecen ganchos para que, mediante herencia o composición, los programadores de aplicaciones definan el comportamiento variable. Algunos patrones de diseño (particularmente el template method y el strategy) son frecuentemente utilizados para introducir esos puntos de extensión.

La construcción y uso de frameworks es una estrategia para modularizar y reutilizar, que mejora la calidad del software y aumenta la productividad de quienes lo hacen. Un framework, visto como módulo de software, no solo encapsula operaciones y estructuras de datos reusables sino que las combina para resolver, con demostrada eficacia, los requerimientos comunes a una familia de aplicaciones. En pocas palabras, encapsula

java.util.logging (un framework de logging en Java)



Logbook usado para el helicóptero Mars Ingenuity de la NASA

1947, Logbook, Computador Mark II

Logging (propósito)

- Agregamos código de login 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!

Herramientas y estrategias

- Podemos utilizar `System.out.println`, pero ...
- Los frameworks/APIs de logging permiten ...
 - Estandarizar la práctica
 - Activar o desactivar logs selectivamente, incluso sin modificar el código
 - Enviar los reportes a distintos formatos y destinos (txt, JSON, XML, archivos, pantalla, sockets ...)
 - Oculta los detalles de implementación

Java Logging Framework

- Framework de logging, incluido en el SDK
- Los loggers se organizan en un espacio jerárquico de nombres
 - Heredan propiedades, y propagan mensajes
- Los mensajes de error se asocian a niveles (importancia)
 - Permite filtrar mensajes por nivel
- Permite enviar mensajes a consola, archivos, sockets
- Permite ajustar el formateo de los mensajes (XML, texto)
- Se puede extender (nuevos formatos, destinos, filtros ...)

Ejemplo básico (1)

```
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");  
    }  
}
```

Ejemplo básico (1)

```
public class Sandbox {  
    public static void main(String[] args) throws IOException {  
        Logger.getLogger("app.main").addHandler(new FileHandler("log.txt"));  
    }  
}
```

La configuración de los loggers suele ocurrir al iniciar la aplicación (y se mantiene globalmente).

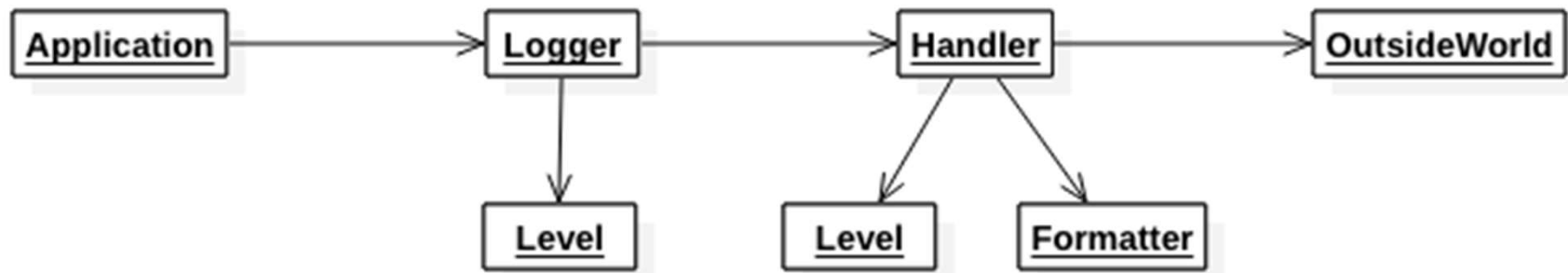
Ejemplo básico (1)

En cualquier método, se puede recuperar y utilizar un logger

```
    } catch (Exception ex) {  
        Logger.getLogger("app.main").log(Level.SEVERE, "Explotó!", ex);  
    }  
    Logger.getLogger("app.main").log(Level.INFO, "App terminada");  
}
```

Partes básicas (visibles)

- Logger: objeto al que le pedimos 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 como se "presentará" el mensaje



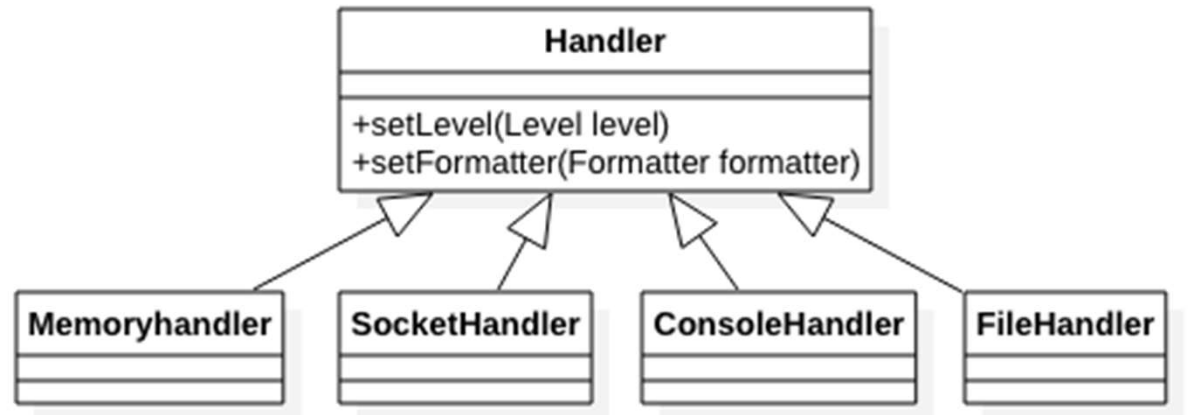
Logger

- 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)
- Envío el mensaje `log(Level, String)` para emita algún mensaje
 - Alternativamente uso `warn()`, `info()`, `severe()` ...

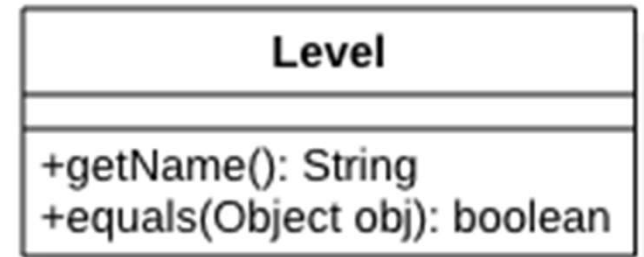
Logger
<ul style="list-style-type: none">+addHandler(Handler handler)+setLevel(Level level)+isLoggable(Level level): boolean+log(Level level, String msg)+warn(String msg)+info(String msg)+severe(String msg)

Handler

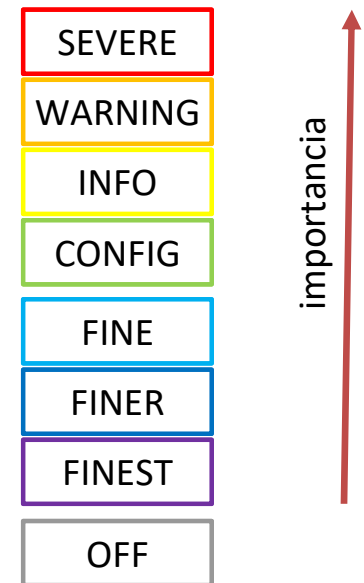
- Recibe los mensajes del Logger y determina como “exportarlos”
- Instancias de MemoryHandler, ConsoleHandler, FileHandler, o SocketHandler
- Puede filtrar por nivel
- Tiene un Formatter



Level

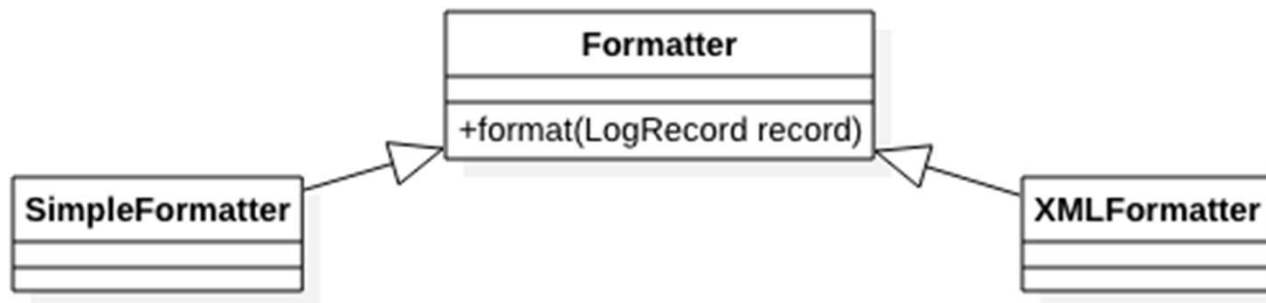


- Representa la importancia de un mensaje
- Cada vez que pido que se loggee algo, debo indicar un nivel
- Los Loggers y Handler comparan el nivel de un cada mensaje con el suyo para decidir si les interesa o no
- Si te interesa un nivel, también te interesan los que son más importantes que ese
- Hay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



Formatter

- El Formatter recibe un 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



Ejemplo avanzado (como caja negra)

//Loggers apagados por defecto

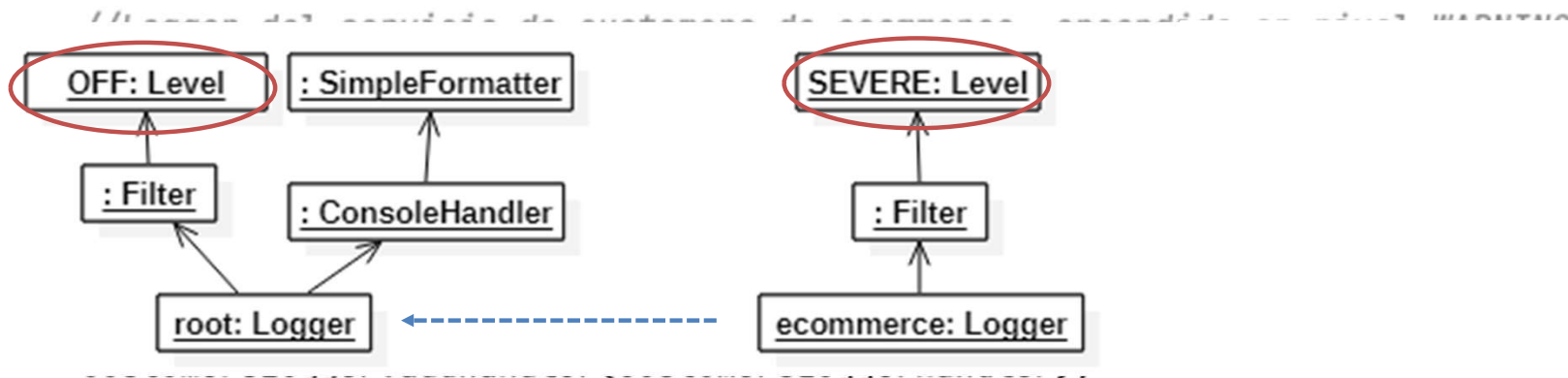
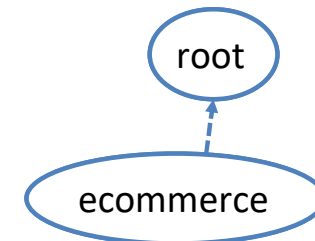
```
Logger.getLogger("").setLevel(Level.OFF);
```

//Loggers encendidos en nivel SEVERE para ecommerce

//Utilizará un ConsoleHandler y un SimpleFormatter

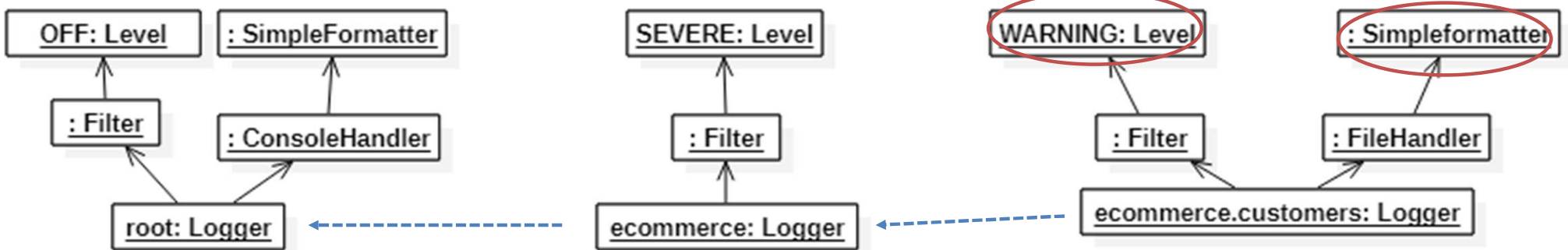
```
Logger rootLogger = Logger.getLogger("ecommerce");
```

```
rootLogger.setLevel(Level.SEVERE);
```



Ejemplo avanzado (como caja negra)

// Loggers agregados por defecto

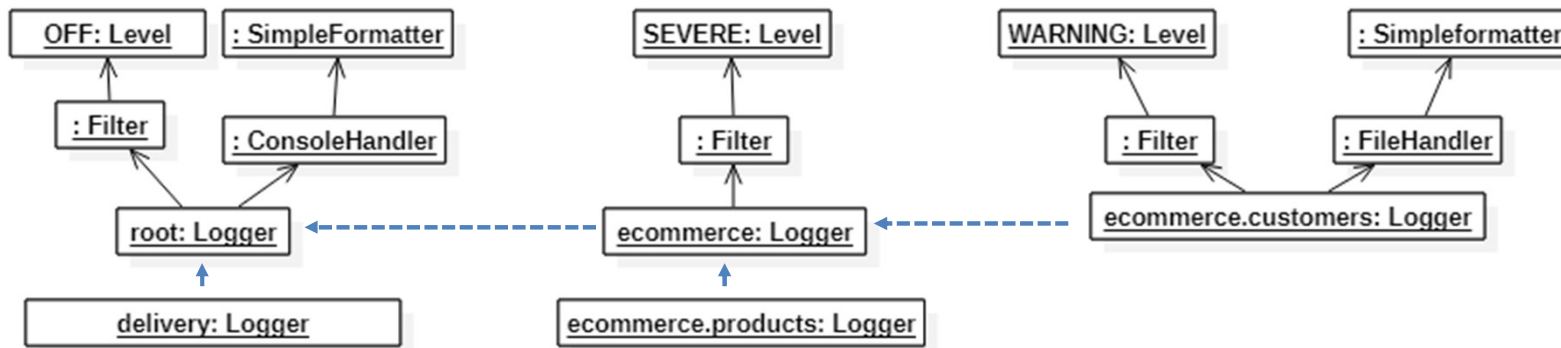


//Logger del servicio de customers de ecommerce, encendido en nivel WARNING

//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 (como caja negra)



// 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");
```

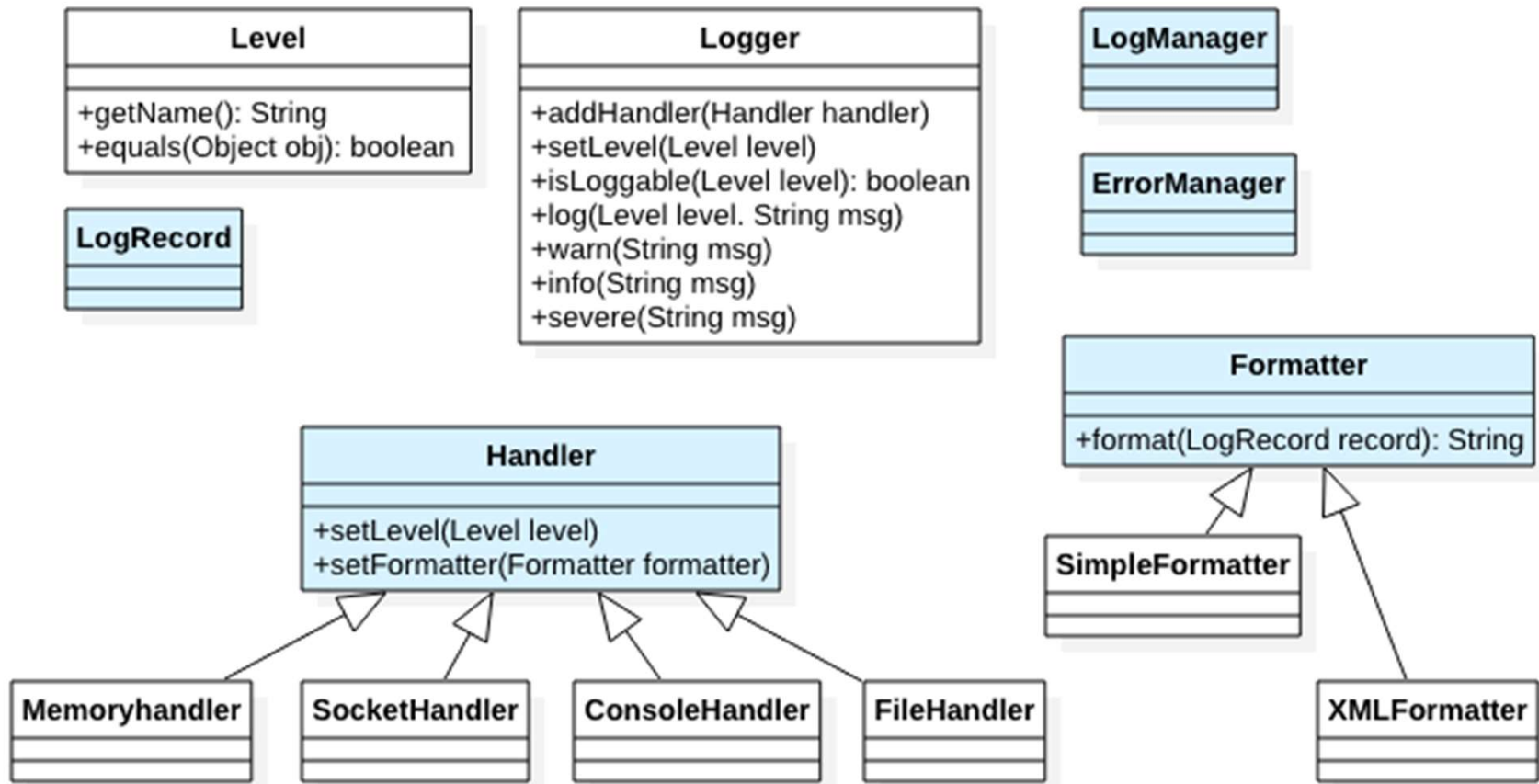
// Este logger hereda de ecommerce por lo tanto ignora el warning

```
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");
```

Lo que no se ve (desde afuera de la caja)



Lo que no se ve (desde afuera de la caja)

- LogRecord: objeto que representa un mensaje
- LogManager: es el objeto que mantiene el árbol de loggers; hay un solo LogManager (global)
- Filter: Logger y Handler tienen un Filter para determinar si algo les interesa o no (el Filter conoce al Level)
- ErrorManager: los Logger pueden tener un ErrorManager para lidiar con errores durante el logging
- Clases abstractas Handler y Formatter
- ... y todas las relaciones entre ellos ...

Lo que no cambia (frozenspot)

- Diseño / propuesta sobre como integrar logging
- Estrategia general
 - Busco un logger para configurarlo o pedirle que emita un mensaje
 - Tengo Logger, Handler, Formatter, Filter, Level
- Cómo se organizan los loggers (árbol gestionado globalmente)
- Las configuraciones se heredan
- Qué pasa cuando le digo log() a un logger (colaboraciones)

Lo que si cambia (hotspots)

- Cuántos y cuales loggers utilizo
- Qué logger “hereda” de cuál
- Qué le interesa a cada logger (su filter)
- Qué handlers se asocian a cada logger
- Qué formatter tiene cada handler y que le interesa (su filter)

Hasta ahí, todo caja negra ...

Extendiendo el framework (caja blanca)

- Y, mirando adentro (caja blanca), 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

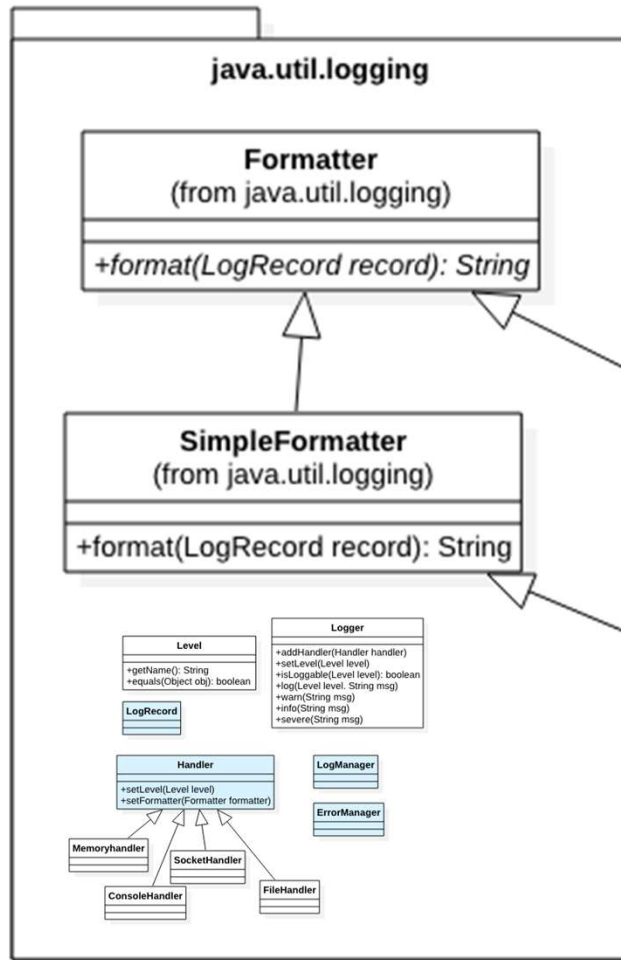
Nuevos Formatters

```
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 "...";
    }
}
```



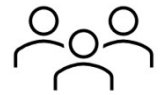
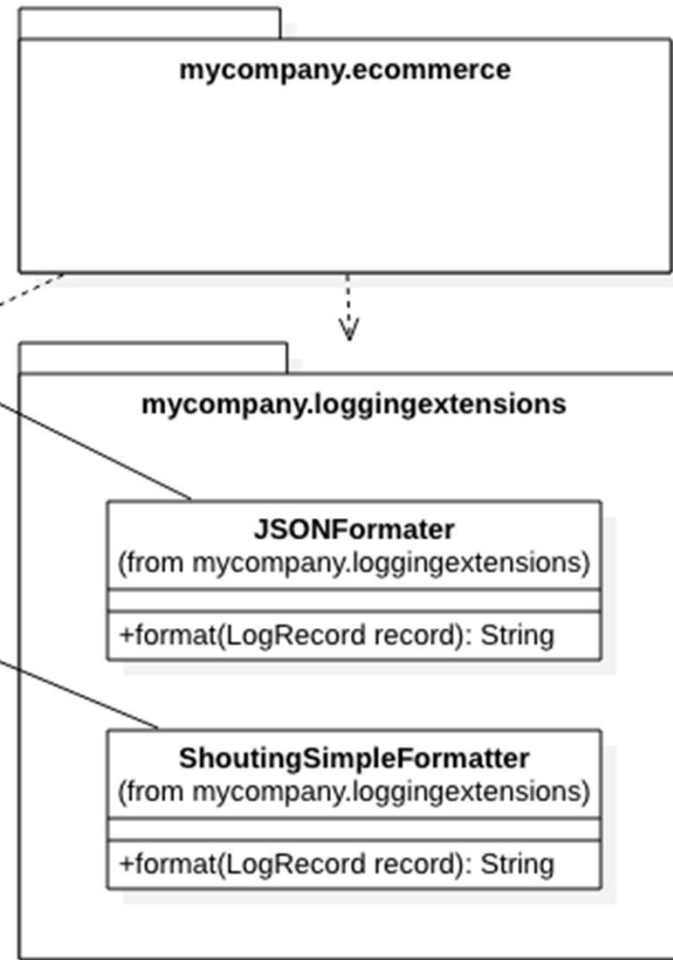
Desarrolladores
del framework



mycompany.ecommerce



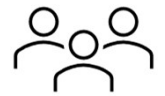
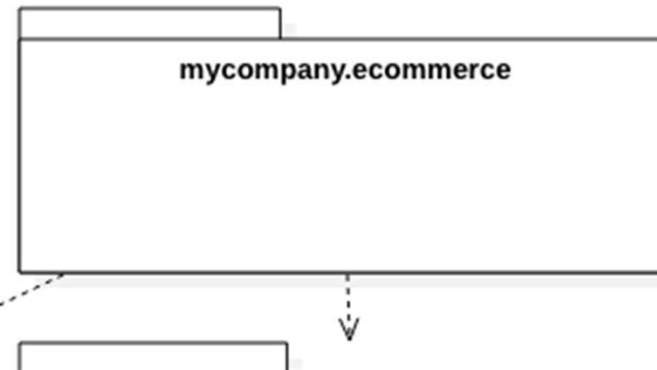
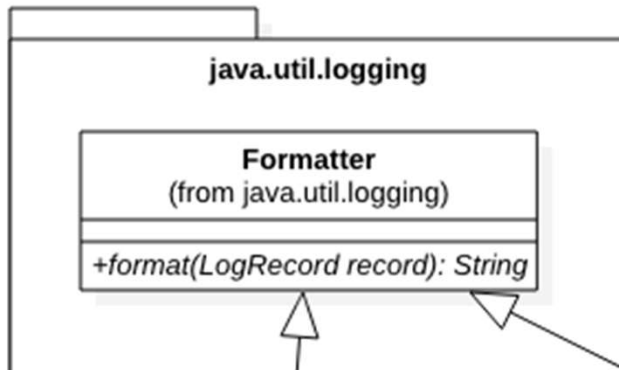
Desarrolladores
de aplicaciones



Mejoradores
del framework

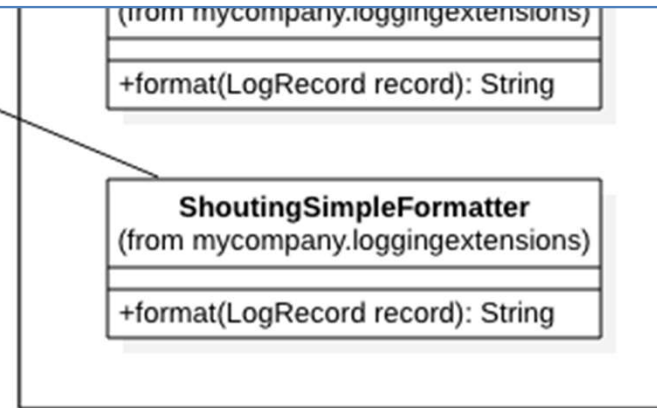
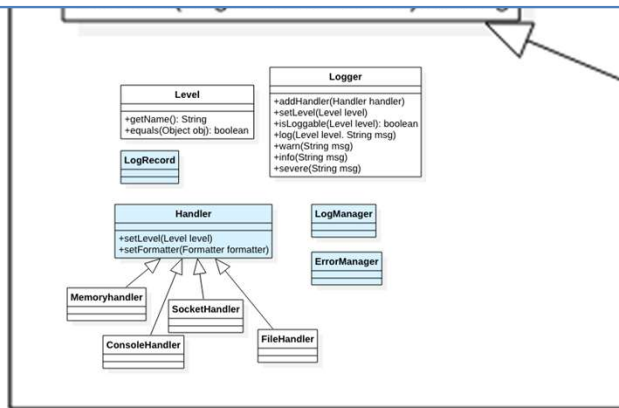


Desarrolladores
del framework



Desarrolladores
de aplicaciones

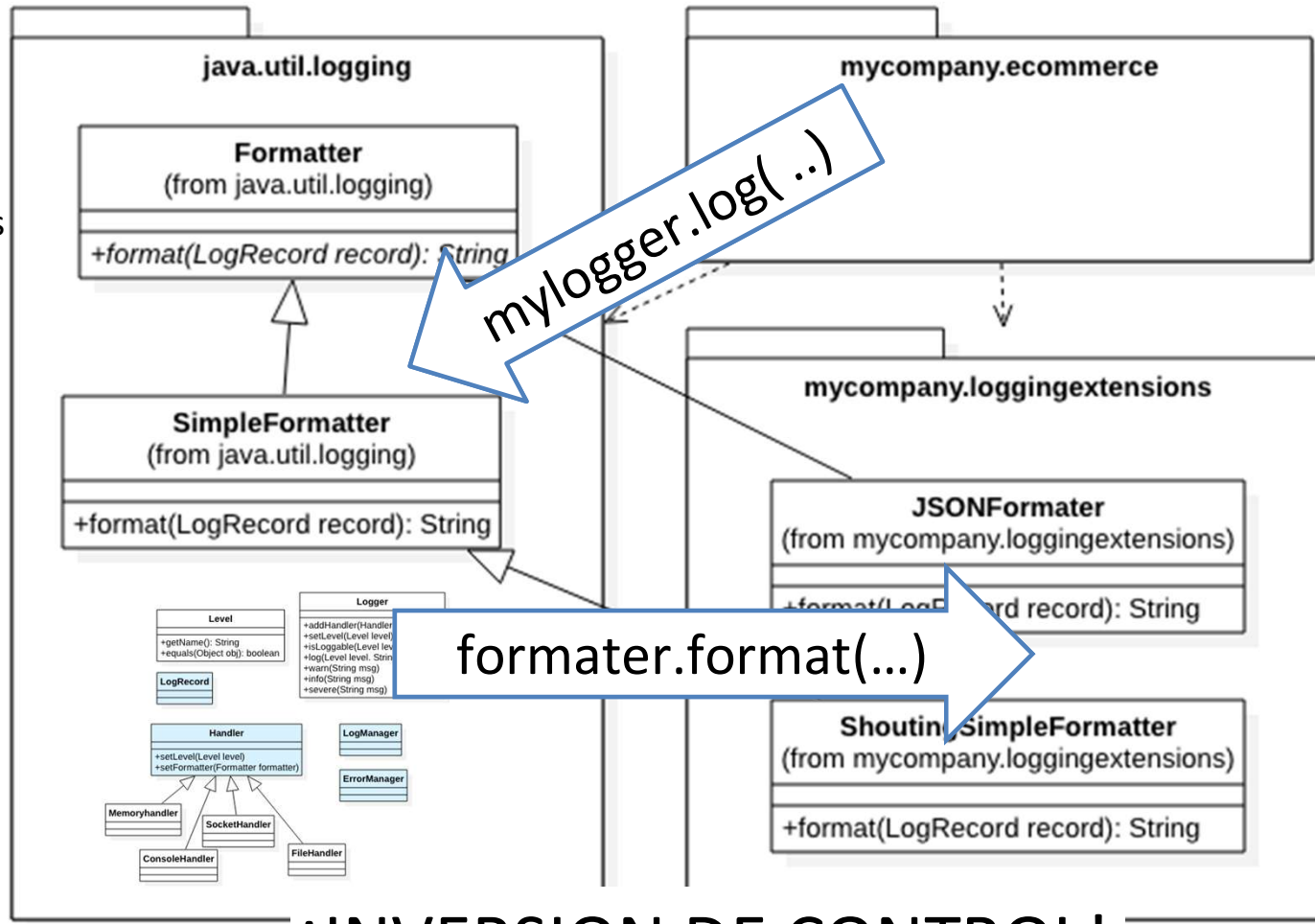
¿Quien envía el mensaje format() a las instancias de nuestras
clases JSONFormatter y ShoutingSimpleFormatter?



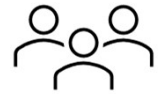
Mejoradores
del framework



Desarrolladores
del framework



¡INVERSION DE CONTROL!



Desarrolladores
de aplicaciones



Mejoradores
del framework

Resumiendo

- `java.util.logging` es un framework que propone una forma de integrar logging en nuestros programas
- Lo utilizo mayormente como una caja negra (instancio y compongo)
- Toma el control cuando le indicamos, y lo devuelve cuando termina
- Puedo extender el framework heredando en los puntos de extensión previstos (caja blanca)
 - En este caso, vamos a percibir la inversión de control

Ejercicios para profundizar

- Armar un árbol de loggers explorando distintas alternativas
- Mirar la documentación y código fuente y reconstruir el diseño
- ¿Encotramos algún patrón en el diseño del framework?
- Escribir nuestro propio handler (¿enviar mensajes por telegram?)
- Escribir nuestro propio formatter (¿JSON?)
- Escribir nuestro propio filter (¿ignorar mensajes severos si no incluyen una excepción?)