Frameworks

REÚSO

el reúso abarca todas aquellas técnicas, procesos y metodologías que tienen como objetivo reutilizar artefactos de software, creados en cualquiera de las etapas, para ser usados en nuevos desarrollos o en la construcción de nuevas versiones

reusamos:

- aplicaciones completas que adaptamos e integramos: Wordpress, GoogleApps, etc.
- componentes y servicios: Twitter API, Facebook API, Google Maps
- funciones y estructuras de datos: de colecciones, fechas, archivos, etc.
- diseños o estrategias de diseño: patrones, algoritmos o estructuras
- conceptos e ideas que funcionan: compartir en las redes sociales, ayuda en contexto

para qué

- para aumentar productividad, reduciendo costos, riesgo, incertidumbre y tiempos de entrega
- para aumentar calidad, porque el reúso aprovecha mejoras y correcciones

dificultades

- problemas de mantenimiento si no tenemos control de los componentes que reusamos
- algunos programadores prefieren hacer todo ellos mismos
- hacer software reusable es más difícil y costoso
- encontrar, aprender a usar y adaptar componentes reusables requiere esfuerzo adicional

algunas alternativas de reúso

- patrones de diseño
- frameworks
- patrones arquitecturales
- software product line: métodos, herramientas y técnicas de ingeniería de software para crear una colección de sistemas de software similares
- librerías: es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca
- program generators: un usuario puede especificar los pasos necesarios para el programa

LIBRERÍAS DE CLASES/ TOOLKITS

una librería de clases es un conjunto de clases que puedo usar independientemente pero todas tienen una relación conceptual

- resuelven problemas comunes de la mayoría de las aplicaciones como manejo de archivos, funciones aritméticas, colecciones o fechas
- cada clase resuelve un problema concreto, independientemente del contexto de uso
- nuestro código controla/ usa a los objetos de las librerías

librerías en Java
codec: implementan algoritmos de encoding/decoding
compress: clases para trabajar con archivos tar, zip, etc.
math: implementan componentes de matemática
java.util.collections: implementan estructuras de datos

FRAMEWORKS

un framework es una aplicación semi completa y reusable que puede ser especializada para producir aplicaciones a medida. O también se define como un conjunto de clases abstractas, relacionadas para proveer una arquitectura reusable para una familia de aplicaciones relacionadas

- proveen una solución reusable para una familia de aplicaciones/ problemas relacionados
- las clases en el framework se relacionan de manera que resuelven la mayor parte del problema en cuestión
- el código del framework controla al nuestro

son el resultado de refactorizar una aplicación, no solo mejora la calidad interna sino abstrae lo que es común en una familia de aplicaciones.

FRAMEWORKS DE INFRAESTRUCTURA

ofrecen una infraestructura portable y eficiente sobre la cual construir una gran variedad de aplicaciones. Algunos enfoques

son interfaces de usuario (desktop, web, móviles), seguridad, contenedores de aplicación, procesamiento de imágenes, procesamiento de lenguaje, comunicaciones

FRAMEWORKS DE INTEGRACIÓN

se utilizan comúnmente para integrar componentes de aplicación distribuidos (la base de datos con la aplicación y ésta con su cliente liviano.). Están 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

FRAMEWORKS DE APLICACIÓN

atacan 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

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.)

INVERSIÓN DE CONTROL

la inversión de control es la característica principal de la arquitectura run-time de un framework. Permite que los pasos comunes de todas las instancias sean especializados por objetos tipo manejadores de eventos a los que invoca el framework como parte de su mecanismo reactivo de despacho

FROZENSPOT vs HOTSPOT

- los frozensports de un framework son aplicaciones construidas con un mismo framework que tienen aspectos que NO podemos cambiar
- los hotspots ofrece puntos de extensión que nos permiten introducir variantes y así construir aplicaciones diferentes

reconocer los hotspots nos permite diseñar mejor el framework y las aplicaciones que lo usan. Instanciar un hotspot puede requerir una combinación de acciones, de herencia y de composición

CAJA BLANCA vs. CAJA NEGRA

los puntos de extensión pueden implementarse en base a herencia o composición. Los frameworks que usan **herencia los llamamos whitebox**, los que usan **composición se llaman blackbox**

COMO USUARIOS DEL FRAMEWORK

HERENCIA	**COMPOSICIÓN**
• implemento, extiendo y redefino métodos • uso variables y métodos heredados • podría cambiar cosas que el desarrollador no tuvo en cuenta • puedo extender el framework • debo aprender qué heredo y qué puedo hacer con ello • no puedo heredar comportamiento de otro lado	• instancio y configuro • conecto a mi código con callbacks • solo conozco algunas clases del framework y sus mensajes • no puedo cambiar o extender el framework • solo tengo acceso a los objetos que recibo de los callbacks

COMO DESARROLLADORES DEL FRAMEWORK

HERENCIA	**COMPOSICIÓN**
• dejo ganchos en clases del framework • paso el control con mensajes a self • no necesito pensar todos los hotspots y todos sus casos • no necesito pasar estado como parámetros • debo documentar claramente qué se puede tocar o no	• dejo objetos configurables para ajustar el comportamiento • paso el control con callbacks • debo pensar todos los hotspots y sus casos • debo pasar/ actualizar todo el estado necesario en los callbacks

• no puedo cambiar el diseño sin preocuparme por los usuarios	• no sé nada del código del usuario • no necesito explicar cómo funciona • puedo cambiar mi diseño sin preocuparme

en resumen, un framework tendrá hotspots que se instancian con herencia y otros que se instancian por composición, por lo general arrancan dependiendo mucho de herencia para ir evolucionando a composición, es más fácil desarrollarlos si son caja blanca y usarlos si son caja negra, es más desafiante desarrollarlos si son caja negra y usarlos si son caja blanca

LOGGING

los log se usa para referirse a la grabación secuencial en un archivo o en una base de datos de todos los acontecimientos que afectan a un proceso particular

los agregamos para entender lo que pasa con ella, por ejemplo para reportes de eventos, errores y excepciones, pasos críticos o inicio y fin de operaciones

resultan útiles para los desarrolladores, administradores y usuarios

no reemplaza al testing

podríamos usar System.out.println pero es mejor incluir los frameworks de logging, los cuales permiten estandarizar la practica, activar/ desactivar logs, enviar reportes en distintos formatos y ocultar detalles de implementación

JAVA LOGGING FRAMEWORK

es un framework incluido en el SDK

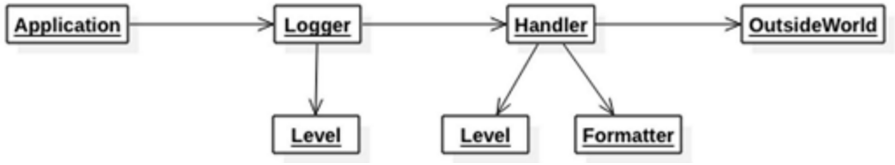
- se organizan en un espacio jerárquico de nombres
- los mensajes de error se asocian a niveles/ importancia
- permite enviar mensajes a consola, archivos y sockets
- es entendible

```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");
    }
}
```

partes básicas

- logger: objeto al que le pedimos que emita un mensaje de log, podemos definir tantos como necesitemos, son instancias de la clase Logger
  - las obtengo con Logger.getLogger(String name)
  - cada uno con su filtro y handler/s
  - se organizan en un árbol a base de sus nombres y heredan configuración de su padre
  - envío el mensaje log(Level, String) para emitir algún mensaje
- handler: encargado de enviar el mensaje a donde corresponda. Recibe los mensajes del Logger y determina cómo "explotarlos", son instancias de MemoryHandler, ConsoleHandler, FileHandler o SocketHandler, puede filtrar por nivel y tiene un formatter
- level: indica la importancia de un mensaje y es lo que mira un Logger y un Handler para ver si le interesa. Representa la importancia de un mensaje, cada vez que pido que se loggee algo, debo indicar un nivel. Los loggers y Handler miran el nivel de un mensaje para decidir si les importa o no
  - los niveles posibles son: Level.SEVERE, Level.WARNING, Level.INFO, Level.CONFIG, Level.FINE, Level.FINER, Level.FINEST (el menos importante), y Level.OFF (nada)
  - si te interesa un nivel, también te interesan los que son más importantes que ese
- formater: determina cómo se presentará el mensaje. Recibe el mensaje de log 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

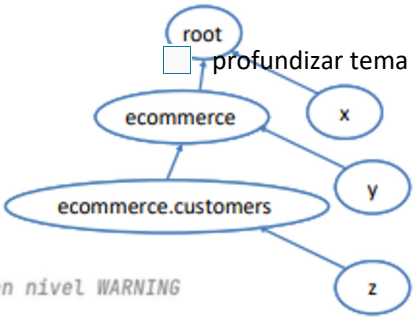


ejemplo

```
//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);

//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);
```



profundizar tema

- getLogger() es un método estático presente en la clase Logger que crea un registrador si no está presente en el sistema con el nombre dado, en este caso "ecommerce"
- FileHandler envía mensajes al archivo que le indiquemos
- SimpleFormatter() para textos de plano simple y no en formato XML
- tanto el Logger como el Hander tienen un level para decidir si están interesados en un determinado logRecord.
- handler puede usar un Formatter para localizar y formar un mensaje
- los niveles asignan la importancia y urgencia de un mensaje de log
- tipos de handler:
  - StreamHandler: un simple handler para escribir registros en un OutputStream
  - ConsoleHandler: handler para escribir registros en System.err
  - FileHandler: handler que escribe registros en un archivo
  - SocketHander: escribe registros en TCP ports
  - MemoryHander: los escribe en memoria
- formatos:
  - SimpleFormatter: escribe resúmenes "entendibles" para las personas
  - XMLFormatter: escribe información estructurada en XML

frozensports:

- diseño/ propuesta sobre cómo integrar logging
- estrategia general, siempre busco un logger para configurarlo o pedirle que emita un mensaje
- tengo logger,handler, formatter, filter y level
- cómo se organizan los loggers (árbol)
- las configuraciones heredadas
- qué pasa cuando le digo log() a un logger

hotsports:

- cuántos y cuáles logreras uso
- qué logger "hereda" cual
- qué le interesa a cada logger

- qué handlers se asocian a cada logger
- qué formatter tiene cada handler y qué le interesa

RESUMEN

java.util.logging es un framework que propone una forma de integrar logging en nuestros programas, lo uso mayormente como una caja negra y toma el control cuando le indicamos y lo devuelve cuando termina. Puedo extender el framework heredando en los puntos de extensión previstos

GUÍA DE LECTURA

un framework puede verse como el resultado de refactorizar una aplicación para mejorar su calidad y para abstraer lo que es común en una familia de aplicaciones similares

este proceso se identifica con lo que se mantiene constante y será responsabilidad de los desarrolladores del framework y lo que puede variar de aplicación a aplicación y es responsabilidad de los desarrolladores de las aplicaciones que usen el framework.

para separar la implementación de la parte constante (a lo que llamamos frozenspots**)** 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 (template y strategy method) son usados para estas extensiones

1) Cuando hablamos de reúso en el marco del desarrollo de software, ¿a qué nos referimos?
   nos referimos a aplicar artefactos, métodos o recursos ya existentes, en un nuevo desarrollo
2) ¿Cuáles son las dificultades que encontramos cuando intentamos reusar código?
   debemos aprender a usarlo/ adaptarlo, a veces preferimos solo usar lo creado por nosotros, problemas de mantenimiento si no hay un control
3) Cuando aplicamos patrones de diseño, ¿Qué estamos reusando? ¿De qué manera los patrones de diseño atacan las dificultades del reúso?
   al aplicar patrones de diseño estamos reusando el esqueleto para resolver un problema recurrente.
4) Cuando utilizamos librerías de clases, ¿Qué estamos reusando? ¿a qué casos apuntan? ¿De qué manera atacan las librerías de clases las dificultades de reúso?
5) Cuando utilizamos frameworks orientados a objetos, ¿Qué estamos reusando? ¿a qué casos apuntan? ¿De qué manera atacan los frameworks las dificultades de reúso?
6) Por lo general, ¿es posible y lógico reusar, de forma aislada, solo algunas clases de un framework? ¿es posible y lógico utilizar solo algunas clases de una librería de clases? Justifique.
7) De acuerdo a el tipo de problema que atacan, ¿Cómo se clasifican los frameworks?
   de clasifican en infraestructura, integración o aplicación
8) ¿A que nos referimos con los términos hotspots y frozenspots?
   - los frozensports de un framework son aplicaciones construidas con un mismo framework que tienen aspectos que NO podemos cambiar
   - los hotspots ofrece puntos de extensión que nos permiten introducir variantes y así construir aplicaciones diferentes
9) ¿A qué nos referimos con "instanciar un framework"? ¿Qué sería "instanciar un hotspots"?
10) ¿En qué se diferencia la visión que tienen de un framework: los desarrolladores del framework, quienes usan/instancian el framework?
    desde la visión de los desarrolladores del framework tendrán como responsabilidad mantenerlo estable, quienes lo usan deberán aprender a usarlo y adaptarlo a su aplicación
11) ¿En qué se diferencia un framework blackbox de uno whitebox? Un framework, ¿cae exclusivamente en una de esas categorías?
    whitebox: usan herencia
    blackbox: usan composición
    sí, los frameworks caen en esas categorías
12) Desde la perspectiva de quien "hace" el framework, ¿Qué implica optar por whitebox, o blackbox? Piense por ejemplo en las facilidades y las dificultades que encuentra en cada caso.
    implicará cómo voy a implementar las variantes, ventajas/ desventajas:

herencia	composición
• simple para casos de pocas alternativas o pocas combinaciones • al implementar los métodos gancho puedo usar variables de instancia y todo el comportamiento heredado de la clase abstracta • si hay muchas variables, empiezo a tener muchas clases y duplicar código porque cada vez que hay una nueva variable, tengo que definir una nueva clase	• evita duplicar código y el creciente número de clases cuando hay varias alternativas porque solo implemento nuevos métodos • puedo cambiar el comportamiento en tiempo de ejecución sin mayor dificultad, caso contrario a herencia que tenía que definir un nuevo objeto

13) Desde la perspectiva de quien "usa" el framework, ¿Qué implica que el mismo sea whitebox, o blackbox? Piense por ejemplo en las facilidades y las dificultades que encuentra en cada caso.
14) La cantidad de "puntos de variabilidad" de un framework (es decir, sus hotspots) depende de cuan complejo es el dominio que ataca (es decir, de cuantas cosas podrían cambiar de aplicación a aplicación). Por lo general, un framework tiene más hotspots que los que el usuario promedio requiere (piense que quien hizo el framework tuvo en cuenta las expectativas de muchos de ellos). ¿Siempre es necesario instanciar (configurar, subclasificar) todos los hotspots o hay algo que el diseñador del framework puede hacer para reducir el trabajo de instanciación del mismo en los casos más simples?
15) ¿Qué patrones de diseño, de los que hemos visto en clase, se encuentran comúnmente en los hotspots de un framework? ó ¿Qué patrones de diseño, de los vistos en clase, son generadores de hotspots?
16) Los conceptos de template y hook (plantilla y ganchos) los encontramos en el template method, en el marco de una relación entre una clase y sus subclases. También se lo puede encontrar en una relación de composición. Ambas situaciones son comunes en los hotspots de un framework. ¿De que manera afecta una y la otra alternativa al comportamiento run time de las aplicaciones que instancia el framework/hotspots?
17) (esto no es un pregunta, es una confirmación) Comprender que un hotspots es en principio un concepto abstracto que describe un acuerdo entre el desarrollador del framework y sus usuarios (esto es claro cuando uno piensa el cosas como los hotspots cards). Puede variar en granularidad. En el código pueden terminar siendo: a) sólo un método; b) clases a subclasificar, implementando un conjunto de métodos en cada una; c) objetos a configurar/componer
18) Le proponen hacer una aplicación en un dominio que es nuevo para usted. Le aclaran que a esa le seguirán muchas parecidas. Usted piensa automáticamente en reúso y en cuanto tiempo y dinero ganará a futuro si puede reusar. Enfoca toda su fuerza en, como primer paso, construir un framework. ¿Tiene sentido? Si cree que si, ¿Cómo lo haría? Si cree que no, cuáles cree que son los pasos para eventualmente poder reusar.
19) ¿Cómo cree usted que se relacionan los temas: patrones de diseño, refactoring (y refactoring to patterns), frameworks, y unit testing?
20)