Librería

- Resuelven problemas comunes a la mayoría de las aplicaciones
- 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
    - Si quiero usar un ArrayList solo me tengo que traer al ArrayList
- Nuestro código controla/usa a los objetos de las librerías
- Algunas librerías de 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

Framework

- Es una aplicación “semicompleta”, “reusable”, que puede ser especializada para producir aplicaciones a medida.
    - Da huecos para que la reutilice.
- Las clases trabajan en un todo
    - 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
- 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 caracteristicas para completar el esqueleto
    - A diferencia de un toolkit o librería, un framework provee una estructura coherente en lugar de un conjunto de “clases útiles”.
- Proveer una solución reusable para una familia de aplicaciones/problemas relacionados
- Es el framework quien te dice como hacer las cosas. El código del framework controla al nuestro. Esto se llama **INVERSION DE CONTROLES**
    - El framework decide cuando pasan las cosas.
- Algunos frameworks de 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.)

Frameworks de infraestructura

- 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, moviles), seguridad, contenedores de aplicación, procesamiento de imágenes, procesamiento de lenguaje, comunicaciones
- Vienen con las plataformas de desarrollo.
- Resuelven problemas de los programadores, no de los usuarios

Frameworks de integración

- Se utilizan comunmente para integrar componentes de aplicación distribuidos
- Están diseñados para aumentar la capacidad de los desarrolladores de modularizar, reusar y extender su infraestrcutura de software para que trabaje transparentemente en un ambiente distribuido

Frameworks de aplicación

- Son frameworks específicos para el dominio.
- Ejemplo: Siu Guarani.

Definiciones

Frozenspots de un framework

- Partes que seran igual para todos los artefactos construídos con el framework
    - Como se obtienen y organizan los Loggers en java.util.logging
    - Cómo/dónde/cuándo indicar que se espera como resultado de cada operación con JUnit
- Todas las aplicaciones construídas con un mismo framework tienen aspectos en común que no podemos cambiar
- Básicamente son cosas que no puedo tocar.

Hotspots de un framework

- Partes del artefacto a construir con el framework, que puedo cambiar
    - Como se formatearán los mensajes de log en java.util.logging
    - Preparación del fixture en tests con JUnit (@BeforeEach)
- El framework ofrece puntos de extensión que nos permiten introducir variantes y así construir aplicaciones diferentes

- Es donde uno puede modificar. Son los “huecos” que va dejando el framework.
- Hay algunos Hotspots que están completados pero puedo cambiarlos e incluso algunos cambios me dejen definir a mi con que quiero llenar.

Instanciando hotspots

- Un framework frecuentemente ofrece muchos hotpost
- Cuando aprovechamos/usamos un hotspot decimos que “instanciamos un hostpot”
- Algunos “debo” instanciar: p.e. Loggers en java.util.logging
- Otros “puedo” instanciar: p.e., Formaters y handlers en java.util.logging
- Instanciar un hotspot puede requerir una combinación acciones, de herencia y de composición
    - Herencia (caja blanca)
        - 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
    - Composición (caja negra)
        - Instancio y configuro
        - Conecto a mi código con callbacks
        - Solo conozco algunas clases del framework y sus mensajes
        - Mis objetos son mis objetos, heredo de donde quiero
        - No puedo cambiar o extender el framework
        - Solo tengo acceso a los objetos que recibo en los callbacks

Caja blanca

- A los frameworks que utilizan herencia y subclasificacion en sus puntos de extensión, les llamamos de Caja Blanca (Whitebox)
- Debo tener un conocimiento del código para poder usarla.
- Que haya caja blanca no quiere decir que me de el código fuente, si no que hay ganchos para que yo haga la herencia o subclasificación de tal manera.
- Que me dé el código fuente no quiere decir que haya caja blanca. Esta bueno verlo para ver que hace.

Caja negra

- A los que utilizan composición les llamamos de Caja Negra (blackbox)
- Tengo que saber poco del código ya hecho.

La mayoría de los frameworks están en algún lugar en el medio. No es blanco y negro. Si hay algo que quiero hacer y no se puede estaría bueno que haya caja blanca. Hacer una caja negra flexible cuesta mas.

Plantillas y ganchos

Las plantillas y los ganchos son una estrategia de programación comúnmente utilizada para introducir puntos de variabilidad (hotspots) en los frameworks orientados a objetos.

Es decir, las plantillas y los ganchos pueden utilizarse para separar la funcionalidad constante (frozenspot) de la funcionalidad variable (hotspots) en un framework. Básicamente las plantillas implementan lo que se mantiene constante, los ganchos lo que varía. Estas pueden implementarse utilizando herencia o composición.

Esto se ocupa de implementarlo el desarrollador del framework

Inversión de control

La inversión de control es una característica clave en la arquitectura de un framework. Permite que los pasos fundamentales del procesamiento de una aplicación, que son comunes a todas las instancias, puedan ser personalizados por objetos llamados manejadores de eventos. Estos manejadores son invocados por el framework de manera automática, como parte de su mecanismo reactivo de despacho.

Hot spots con herencia

- El template method asegura que se cumplan los pasos
- Programamos “por diferencia” implementando los hooks, utilizando herencia en donde la plantilla se implementa en una clase abstracta y los ganchos en sus subclases .
- Tenemos acceso a las V.I. y métodos que heredamos (muy flexible)
- La jerarquía se explota si hay muchas combinaciones
- No podemos cambiar el comportamiento en run-time
- Como desarrollador del framework:
    - Dejo ganchos en clases del framework (abstractas y concretas)
    - Paso el control con mensajes a self
    - Se que esperar del código del usuario (si hace lo que digo)
    - No necesito pensar todos los hotspots y todos sus casos
    - No necesito pasar estado como parámetros (está en las v.i.)
    - Debo documentar claramente que se puede tocar y que no
    - No puedo cambiar el diseño sin preocuparme por los usuarios

Hotspots con composición

- El template method asegura que se cumplan los pasos
- Implementamos los hooks, utilizando composición en donde un objeto implementa la plantilla y delega la implementación de los ganchos en sus partes.
- NO tenemos acceso a las V.I. ni métodos privados ni de los otros componentes.
- Dependemos de las opciones provistas
- “Podríamos” sub-clasificar las componentes
- Como desarrollador del framework:
    - 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 se nada del código del usuario
    - No necesito explicar (mucho) como funciona
    - Puedo cambiar mi diseño sin preocuparme (mucho) por los usuarios

Un framework tendrá hotspots que se instancian con herencia y otros que se instancian por composición

Por lo general, arrancan dependiendo mucho de herencia (caja blanca) para ir evolucionando a composición (caja negra)

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