Frameworks

¿Qué reusamos? De todo → Conceptos e ideas que funcionan, patrones de diseño, algoritmos de búsqueda, arquitecturas, funciones, estructuras de datos, aplicaciones completas que adaptamos e integramos (wordpress, drupal, ...)

¿Por qué reusamos?

- Para aumentar la productividad
- Para aumentar la calidad

¿Qué dificultades tiene el reuso?

- Problemas de mantenimiento si no tenemos control de los componente que reusamos (que algo se deje de actualizar por ejemplo)
- Hacer software reusable es mas difícil y costoso (paga a la larga)
- Encontrar, aprender a usar y adaptar componentes reusables requiere esfuerzo adicional (capacitarse requiere tiempo)

¿Qué estrategias hay finalmente para el reuso?

- Patrones de diseño...
- Frameworks

Tanto las librerias como los frameworks, en el contexto de la materia, son codigo, son clases de objetos

Librerias 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

Frameworks

- Un framework es una aplicación “semicompleta”, “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...
- 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” ...

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 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, moviles), seguridad, contenedores de aplicación, procesamiento de imágenes, procesamiento de lenguaje, comunicaciónes
- 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 aplicacion (o Enterprise)

- Estos frameworks atancan 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 RelationshipManagement) 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 entreprise puede encapsular el conocimiento y experiencia de muchos años de una empresa, tranformandose en la clave de su ventaja competitiva y su mas preciado capital
- Son la base para la implementación de líneas de productos (SPL)

FrozenSpot vs Hotspot

Frozenspot

Los *frozen spots* son elementos del framework que no se pueden modificar.
Como resultado, todas las aplicaciones creadas con el mismo framework compartirán ciertas características.

- Aspectos del framework que no podemos cambiar ni modificar
- Estos aspectos o frozenspots generan que distitnas aplicaciones hechas con el mismo framework tengan aspectos en comun o caracteristicas similares
- **Ejemplos**

Hotspot

Por otro lado, los **hotspot son puntos de extensión que ofrece el framework que nos permiten introducir variantes** y asi construir aplicaciones diferentes

- Los puntos de extensión pueden implementarse en base a herencia o en base a composición

- Los hotspots o puntos de extension son puntos específicos en la estructura del framework que permiten introducir variantes, para asi construir aplicaciones diferentes
- Es todo aquel aspecto que puedo extender de un framework
- Los puntos de extensión pueden implementarse en base a herencia o en base a composición
- **Ejemplos**

IMPORTANTE : Cuando hablamos de frozen spot y hotspot no estamos hablando de codigo, sino de funcionalidad, habra funcionalidad que el framework me deje extender o personalizar y otra que no

- No tiene sentido decir, “esta clase es un hotspots”
- **Los hotspots y los frozenspots son algo que se describe/identifica a nivel de especificación de requerimientos**
- Ejemplo de especificaciones de un hotspot: Quiero que tal cosa sea cambiable o modificable, que tenga tales opciones, etc
- Despues se verá qué tenicas uso para que en mi codigo, esas especificaciones se transformen en un hotspot

Ejemplos hotspots y frozenspots

Ejemplo 1: Imagina que estás diseñando un framework para construir aplicaciones de comercio electrónico. Durante la especificación de requerimientos, decides lo siguiente:

- **Hotspots (puntos de extensión)**:
  - **Carrito de compras**: Debe ser personalizable para que los desarrolladores puedan agregar diferentes reglas de descuento, impuestos o métodos de pago.

    - **Catálogo de productos**: Debe permitir que los desarrolladores definan cómo se organizan y filtran los productos (por categoría, precio, etc.)
- **Frozenspots (partes fijas)**:
    - **Autenticación de usuarios**: Debe ser estándar y no modificable para garantizar la seguridad y consistencia en todas las aplicaciones.
    - **Manejo de transacciones**: Debe ser fijo para asegurar que todas las transacciones se procesen de manera confiable y segura.

EJEMPLO 2 : Framework de desarrollo de juegos

Especificación de requerimientos:

- **Hotspot**: El sistema de físicas debe ser personalizable para que los desarrolladores puedan ajustar la gravedad, fricción, etc., según el tipo de juego. Ademas deben poder introducir sus propias fisicas (colisiones, parabolas, etc)
- **Frozenspot**: El motor de renderizado debe ser fijo para garantizar un rendimiento óptimo y consistente en todos los juegos.

EJEMPLO 3 :

Robot wars - Framework

Frozen spot

- En cada ronda, se activan los robots uno a uno - reciben el mensaje **step()**
- Ejecutan las siguientes operaciones en orden:
  - Avanzar
  - Consumir bateria
  - Disparar
  - Recolectar artefactos

Hot spots

- Varias formas de avanzar:
  - Orugas, Ruedas, Overcraft
- Varios tipos de baterias:
  - Cadmio, Nuclear, Solar
- Varios equipos de armamento:
  - Laser, Bombas, …

- Un framework ya sea del tipo que sea nos resuelve diversos aspectos de los que nosotros como usuarios del software no nos tenemos que preocupar
- Pero posiblemente haya funciones que el framwork no implemente como tal, o que dependan del dominio en el que ese framework esta trabajando
- Cuando este no implementa una funcionalidad, probablemente deje alguna forma de exteneder la funcion del framework, a eso se le denomina **puntos de extension**
- En cambio, habra cosas que el framework no deja sujeto a modificacion, por ejemplo, **la forma de organizar informacion en una base de datos** o la forma en la que nuestra informacion debe estructurarse, esas cosas que el framework no deja lugar a modificacion se le denominan frozen spot

Frameworks de Caja blanca o Caja negra

- 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
- La mayoría de los frameworks están en algún lugar en el medio :
  - Algunos usuarios los ven como caja negra
  - Otros como caja blanca

Framework de caja blanca

- A los frameworks que utilizan **herencia** en sus puntos de extensión, les llamamos de Caja Blanca (Whitebox)
- Y en el otro extremo tenemos frameworks que para poder hacer una aplicacion debemos entender cómo se conectan las piezas, como intectuan (probalemente deba subclasificar, entender cuando se reciben los mensajes, qué se puede hacer dentro de los metodos, etc) y de alguna manera se trata de entender el framework y terminar de completarlo. Es como si nos dijeran “tenemos la app casi lista, entendela y completa lo que falta”

Framework de caja negra

- A los que frameworks que utilizan **composición** les llamamos de Caja Negra (blackbox)
- Hay frameworks que se pueden usar, instanciando objetos y componiendolos (sin entender como esas cosas funcionan o quien le manda mensajes a

quien). Esto puede llevarse al extremo en donde uno ni siquiera escribe codigo sino que uno setea cosas simplemente (frameworks muy maduros)

Los frameworks tienden a pasar de ser de caja blanca a caja negra

Herencia vs Composición

Herencia

La herencia es cuando un objeto hereda estado y comportamiento de otro. En el caso de las interfaces, no hay herencia, sino que las subclases de una interfaz **implementan** la interfaz, pero no heredan comportamiento alguno.

- La herencia es más estatica y no puede ser facilmente cambiada en tiempo de ejecución

Cuando uso herencia como usuario de un framework podría cambiar cosas que el desarrollador no tuvo en cuenta, puedo extender el framework, uso variables y métodos heredados, Implemento, extiendo, y redefino métodos

Composición

Es una relacion en la que un objeto depende de otro. No puede funcionar sin dicho objeto

- La composición puede cambiar en tiempo de ejecución

Cuando uso composicion como usuario de un framework simplemente instancio y configuro, conecto a mi código con callbacks (paso funciones como parametros), y no puedo cambiar o extender el framework

Inversión de control

La arquitectura en tiempo de ejecucion de un framework se caracteriza por la **inversion de contro**

- Este principio de diseño permite que el flujo de control del programa sea invertido : En lugar de que el código de la aplicación controle el flujo de la ejecución, un framework o contenedor externo toma el control y llama a los componentes según sea necesario

La inversion de control permite al framework determinar qué set de métodos especificos de la aplicacion invocar en respuesta a eventos externos

- El codigo que escriben los programadores de un framework **le envia mensajes** a las clases escritas por mi
- Lo normal seria que las clases escritas por mi nunca sean instanciadas, asi como los métodos implementados en dichas clases, nunca los ejecutare desde mi codigo. A esas clases y esos mensajes los instancia y los envia **el propio framework**
- Por eso se dice, que el codigo del framework invoca a mi codigo. A esto se le llama **Inversion de control.** El codigo del framework controla el nuestro.

En mis palabras : La inversión de control es un mecanismo que poseen los frameworks que les permite ejecutar codigo del desarrollador ante determinados eventos, en lugar de que el codigo del desarrollador invoque al codigo del framework

Ejemplo: Web Framework con Manejo de Ruta

Imagina un **framework web** como Express.js (Node.js) o Spring Boot (Java). En lugar de que tu código controle el ciclo de vida de las solicitudes HTTP (ejecutando un bucle para escuchar peticiones), el *framework* maneja ese flujo y delega en tu código solo cuando es necesario.

Sin Inversión de Control (Enfoque Tradicional):

```
// Pseudocódigo tradicional (sin IoC)
const server = createServer();
server.listen(3000);

while (true) {
  const request = waitForRequest();
  if (request.path === "/saludo") {
    enviarRespuesta("Hola mundo");
  } else {
    enviarRespuesta("Ruta no encontrada");
  }
}
```

Con Inversión de Control (Usando un Framework): El framework controla el bucle de eventos y **llama a tu código** cuando ocurre una acción relevante (ej: una ruta solicitada):

```
// Ejemplo en Express.js (Node.js)
const express = require('express');
const app = express();

// Tú defines las rutas, pero el framework invoca tu función cuando llega una soli
app.get('/saludo', (req, res) ⇒ {
  res.send('Hola mundo');
});
```

```
app.listen(3000); // El framework maneja el bucle de escucha
```

¿Dónde está la inversión?

- **Sin IoC**: Tú controlas el flujo (ej: el bucle `while`).
- **Con IoC**: El framework controla el flujo y **invoca tus funciones** cuando se disparan eventos (rutas, clics, etc.). Tú te limitas a definir la lógica específica.

Quien escribe un framework es como si estuviera escribiendo una aplicacion pero que en algun momento deberá continuar otra persona. Es decir, quien hace un framework debe parar y pensar que **alguien** podria querer "continuar" el codigo o extenderlo en algun punto especifico para implementar sus propias funciones. Quien hace el framework deja "ganchos" donde alguien podria engancharse para intentar agregar una variante

Como diseñador del framework determino :

- Qué cosas van a ser iguales para todas las aplicaciones hechas con mi framework
- Qué cosas van a poder variar