Resumiendo Frameworks

Frozen Spots:

Son partes fijas del framework. Secciones que no puede ni debe modificarse, porque forman parte de la logica interna y del control principal del framework.
En la mayoria de los casos, los frozenSpot se encuentran dentro del loop de control. Los frozen spot representan decisiones tomadas por los desarrolladores del framework, que no esperan cambios.
Por ejemplo la forma en la que el framework invoca metodos o maneja eventos.

HotSpot:

Punto donde se espera que el desarrollador o el que usa el framework, que pueda agregar o cambiar el comportamiento.
Por ejemplo implementando una interfaz o extendiendo una clase abstracta.

Hook:

Punto donde el framework llama a un comportamiento que nosotros definimos.
El framework lo invoca automaticamente (por ejemplo un metodo onStart())

Comparacion entre HotSpot y Hook:

Ambos permiten redefinir comportamiento, es decir, tanto los hooks como los hotspot son puntos de extension del framework.

Framework de caja negra vs caja blanca:

Framework de caja nega: Es aquel donde no se ve ni se puede modificar el loop de control. Simplemente se siguen las reglas para utilizar el framework y definimos lo que nos permitan.
En este tipo de frameworks **no sabemos cuando ni como se llama el codigo.**
No sabemos ni modificamos el flujo interno, solo podemos "enchufar" componentes.

Framework de caja blanca: Tenemos visibilidad y controil sobre el flujo del framework (loop de control), por ejemplo podemos extender clases del framework y redefinir metodos.

Que es lo que distingue un framework de una libreria?

Una libreria la podemos utilizar para obtener instancias de clases y llamar a sus metodos. Es decir, llamamos a sus clases para obtener algo.
Una libreria, *NO DEFINE EL FLUJO DEL PROGRAMA*

- Una libreria la usamos directamente, por ejemplo LocalDate.now

Un framework lo podemos utilizar para ejecutar una familia de aplicaciones. Aca nosotros no llamamos al codigo, sino llamamos al framework el cual llama a nuestro codigo. Se basan en el principio de inversion de control
Un framework define como se estructura nuestra aplicacion

Plantillas y ganchos con herencia y composicion

Las plantillas y los ganchos son una estrategia de programacion utilizada para introducir puntos de variabilidad (hotspots) en los frameworks orientados a objetos.
Los ganchos pueden utilizarse usando herencia o composicion.

Cuando tenemos herencia y cuando composicion?

Vamos a recordar primero ambas definiciones:

Herencia:
Es un mecanismo mediante una clase adquiere (hereda) los atributos y metodos de otra clase.

- La clase que hereda se la conoce como subclase.
- La clase base a la cual heredan se la conoce como superclase.

```
abstract class Animal{
 protected String nombre;
 public Animal (String unNombre)
 {
     this.nombre = unNombre;
 }

 public abstract void hacerRuido();

 public Strin getNombre()
 {
    return this.nombre;
 }
}

class Perro extends Animal {

    public Perro(String nombre)
    {
      super(nombre);
    }
```

```
    @Override
    public void hacerRuido()
    {
        System.out.println("guaffff");
    }

}
```

En este caso, se utiliza herencia, la clase Perro seria una subclase de Animal, y la subclase redefine el metodo hacerRuido, ademas, hereda de su superclase la variable de instancia nombre y el metodo getNombre.

Composicion:
Consiste en construir objetos mas complejos utilizando otros objetos en lugar de heredar una clase.

- Se usa mucho con interfaces o ganchos (hooks) para delegar comportamientos.

```
class Motor {
    void encender() {
        System.out.println("Motor encendido");
    }
}

class Auto {
    private Motor motor = new Motor();

    void arrancar() {
        motor.encender();
    }
}
```

`Auto` **compone** un `Motor`. No hereda de él, sino que lo usa.

🎯 ¿Cómo detectar si un framework usa herencia o composición para puntos de extensión?

🔵 Señales de Herencia

- Te pide extender una clase base para añadir o modificar funcionalidad.
- Usa `abstract` o `protected` para que sobrescribas métodos.

🟢 Señales de Composición (Ganchos, Hooks, Interfaces)

- Te pide **inyectar dependencias**, **registrar callbacks**, o pasar objetos que implementan ciertas interfaces.
- Usás interfaces como `Runnable`, `Command`, `EventListener`, etc.

🧭 ¿Cómo saber qué usar?

Situación	Preferible
Necesitás cambiar el comportamiento interno	Composición
Tenés una jerarquía clara (Ej: `Empleado -> Gerente`)	Herencia
Querés reutilizar funcionalidad sin acoplarte	Composición
Tenés que implementar múltiples variantes	Composición
Solo necesitás sobrescribir uno o dos métodos simples	Herencia puede servir

Como darnos cuenta si se usa herencia o composicion:

Herencia: Lo importante de la herencia es que el comportamiento principal se defina o se deje abierto para redefinirlo en subclases, y NO que invoque a metodos de otras clases

- Las caracteristicas de la herencia son:
  - Permitir extender la clase base.
  - Sobreescribir metodos para cambiar el comportamiento.
  - No se necesitan inyectar objetos.
  Composicion: Utiliza una clase mas compleja con una variable de instancia que invoca a metodos especificos. Esa variable de instancia representa la composicion (comportamiento delegado). Nos vamos a dar cuenta porque no deja metodos abiertos a posibles extensiones, sino que utiliza otras clases externas. Estan orientados a usar comportamiento existente de otras clases, y no tanto a redefinir o crear nuevos comportamientos.

🎯 Conclusión

> Cuando un framework tiene una clase abstracta, **podría usar herencia o composición** como mecanismo de extensión:

- Si usa **herencia**, lo esperable es que crees subclases que sobrescriben métodos para definir o cambiar el comportamiento.

- Si usa **composición**, es probable que la clase abstracta (o base) tenga **atributos que son interfaces u objetos** que delegan ese comportamiento. En ese caso, **configurás el comportamiento inyectando diferentes objetos**, no por heredar.