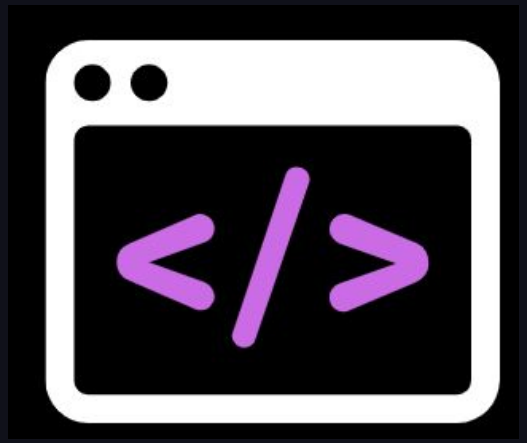


Exposición Teórica

Programación Orientada a Aspectos

<Edgar Daniel Gonzalez, Miguel Angel Puentes>

<Jhonatan Steven Rodriguez, Paula Daniela Velosa>

{Contenido}

Motivación	¿De qué problema surge la Programación Orientada a Aspectos?
Filosofia del Paradigma	¿Qué es Programación Orientada a Aspectos (POA)?
Conceptos Clave	Conceptos necesarios para aplicar Programación Orientada a Aspectos
Ventajas y desventajas	Lo mejor y lo que puede mejorar del paradigma
Lenguajes de Programación	Lenguajes de programación que utilizan POA

{Contenido}

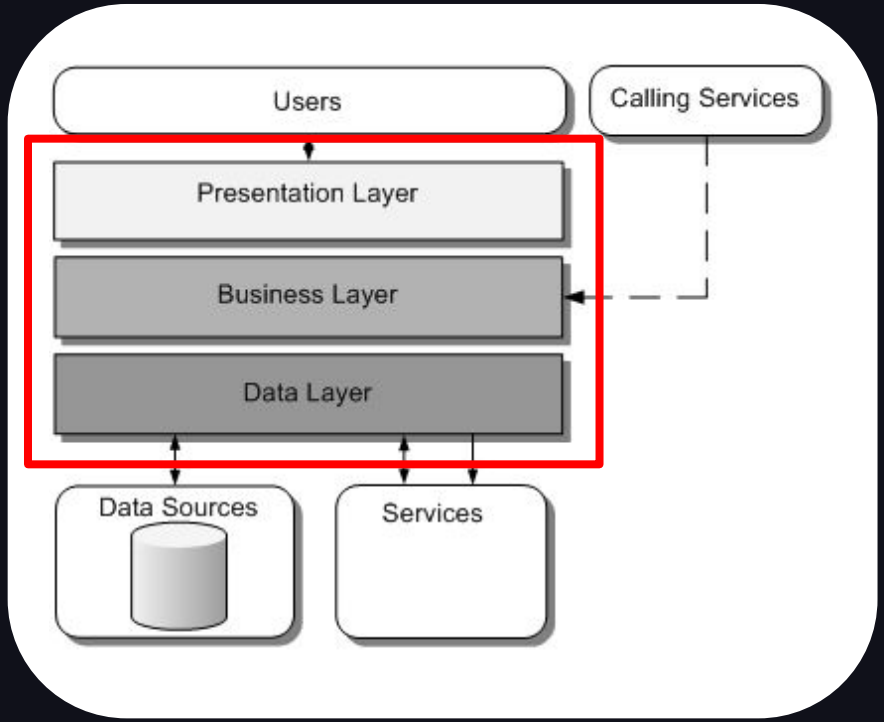
Ejemplos	Ejemplos utilizando Programación Orientada a Aspectos
Aplicaciones	¿Cuándo es mejor usar POA?
Referencias/Bibliografía	Material de referencia y consulta

{Motivación}

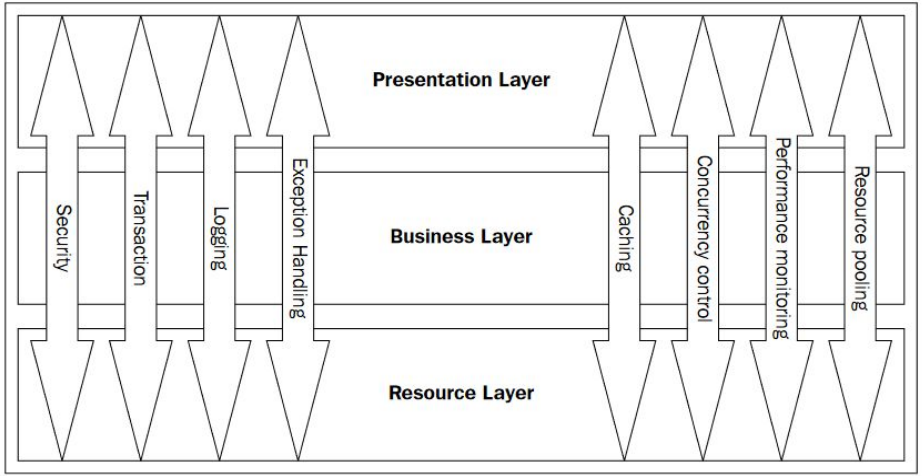
En POO la estructura del código luce, más o menos, de la siguiente forma:

Se ve organizado, ¿no? Y la estructura sugiere una buena separación de responsabilidades. ¿Cuál es el problema?

Veamos…



{Motivación}

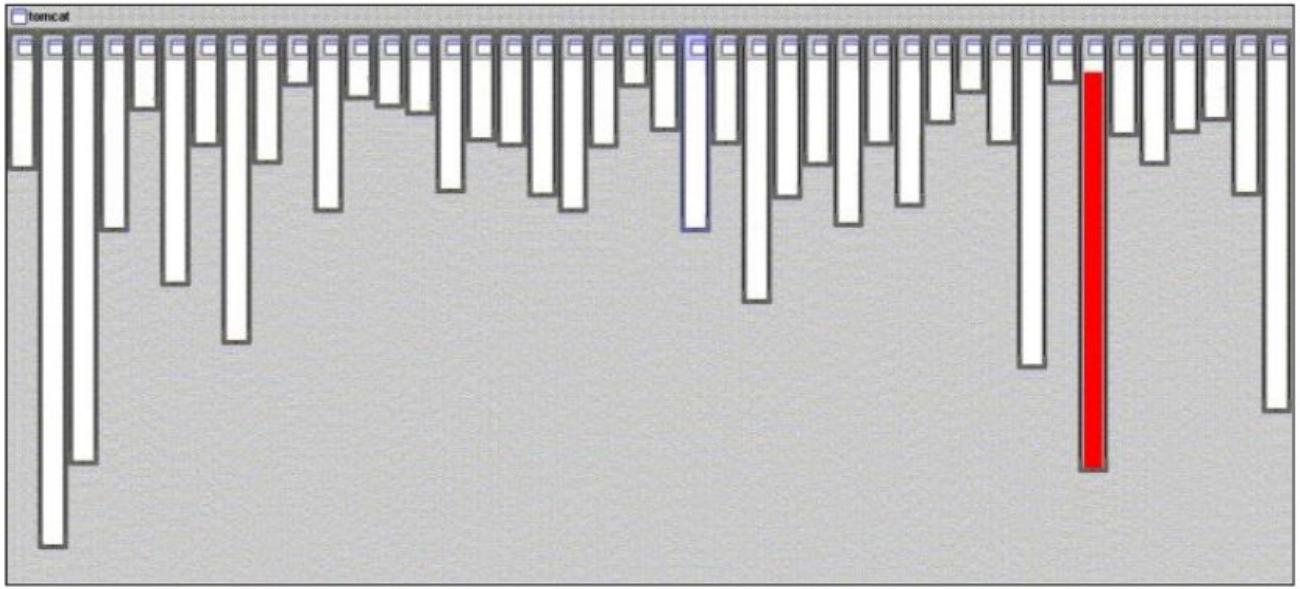


Hay funcionalidades que son transversales (*cross-cutting concerns*).

¿Qué problemas tiene esto?

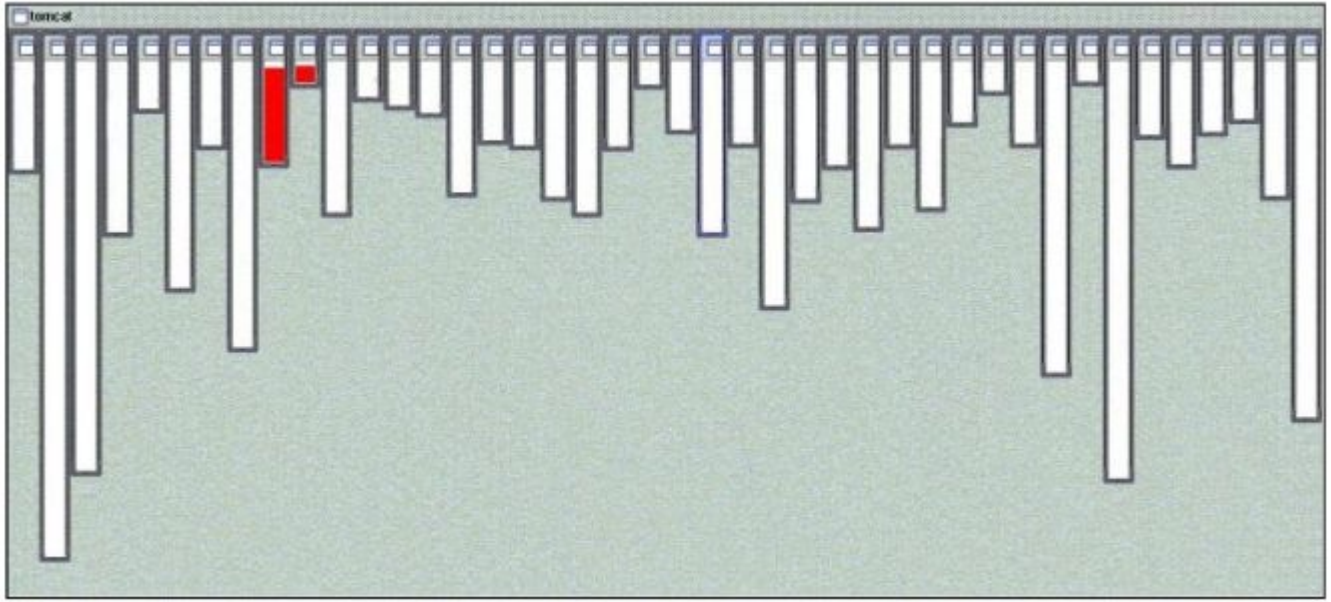
1. Hace el código más difícil de entender.
2. Se tienen que adaptar varias partes del sistema.
3. Depuración más complicada
4. ¡Codigo repetido!

{Motivación}



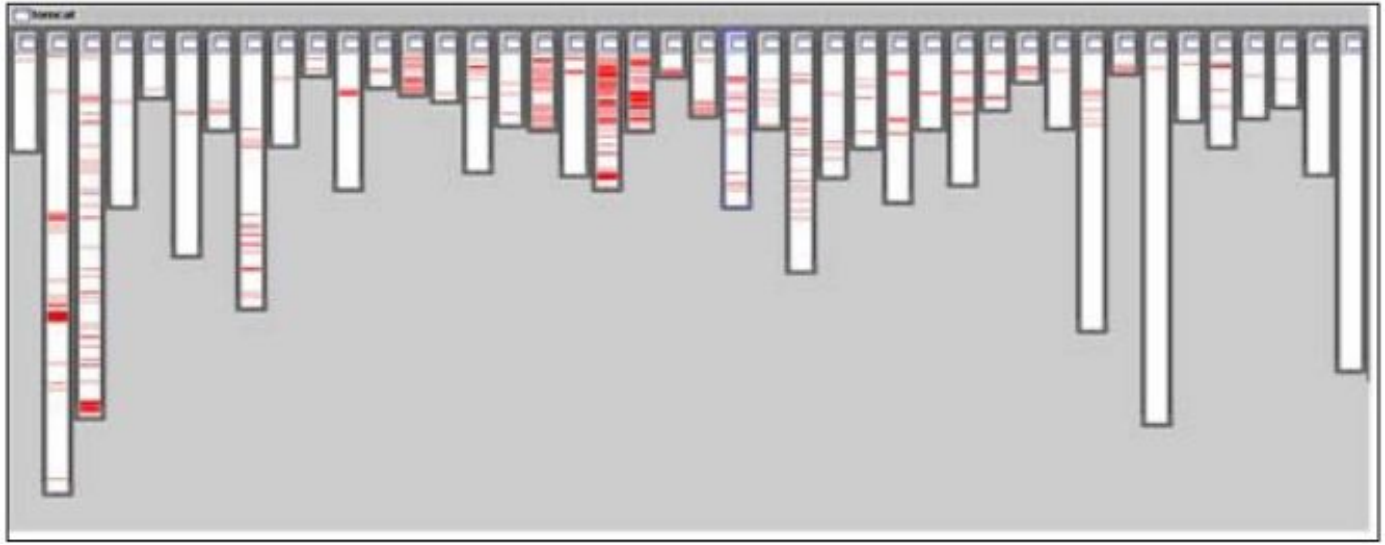
The code of **Servlet Engine Tomcat 4** divided in modules

{Motivación}



The code of **Servlet Engine Tomcat 4** divided in modules

{Motivación}



The code of **Servlet Engine Tomcat 4** divided in modules

{Filosofia del paradigma}

<¿Qué es Programación Orientada a Aspectos (POA)?>

{Generalidades}

01 <Debate>

Existe un debate acerca de si es o no un paradigma, en esta exposición se asume que es un paradigma.

02 <Relación con POO>

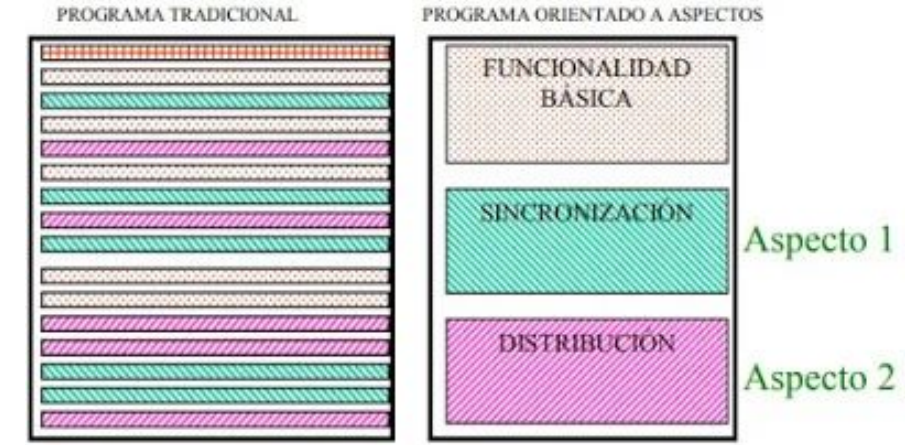
Este paradigma tiene una fuerte relación con la programación orientada a objetos. Muchas personas afirman que es una extensión de POO.

<¡Veamos de qué se trata!>

{Filosofia POA}

El paradigma busca permitir una mayor modularidad, flexibilidad y orden.

Los programas se pueden separar en Aspectos y Componentes.



<Objetivo>: Proporcionar un conjunto de principios y conceptos que permita diseñar y desarrollar software.

{Conceptos Clave}

<Conceptos necesarios para aplicar Programación Orientada a Aspectos >

{Conceptos Clave}

<Concepto>: Es un requerimiento que debe ser implementado en el sistema.

<Componente>: Conjunto de funcionalidades de un sistema encapsuladas.

<Aspecto>: Encapsula una preocupación transversal que puede ser aplicada en ciertos puntos.

{Conceptos Clave}

<Consejo>: ejecuta un aspecto.

<Punto de enlace>: puntos en código donde se implementa un consejo

<Punto de corte>: Expresiones que seleccionan puntos de enlace.

{Conceptos Clave}

<Consejo>: ejecuta un aspecto.

Before advice
After advice
After throwing advice
Around advice

<Punto de enlace>: puntos en código donde se implementa un consejo

{Conceptos Clave}

<Consejo>: ejecuta un aspecto.

<Punto de enlace>: puntos en código donde se implementa un consejo

<Punto de corte>: Expresiones que seleccionan puntos de enlace.

{Conceptos Clave}

<Punto de enlace>

- Ejecución de un método
- Manejo de una excepción

<Punto de corte>

- execution(<paquete de métodos>..*(..))*
- execution(com.example.myapp..*(..)) throws Exception)*

<Función>: Unidad de código a la cual se le aplica aspectos, es transversal al sistema. {Principal diferencia con POO}

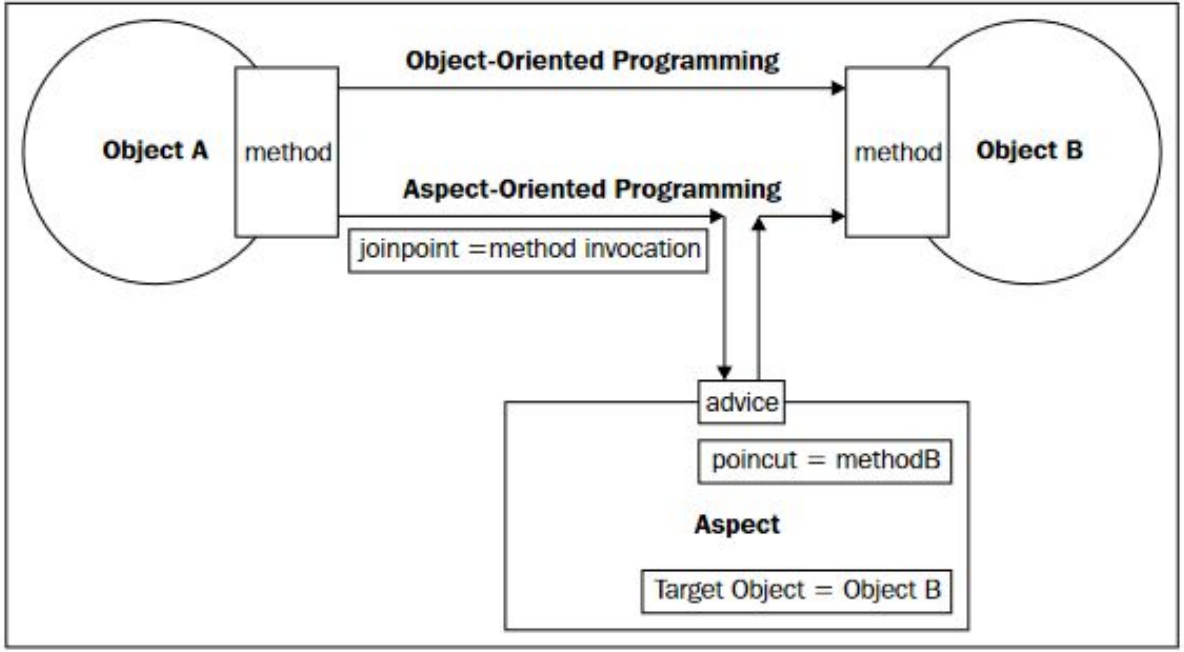
{Conceptos Clave}

<Tejedor>: Aplica los consejos en los puntos de unión, puede darse en tiempo de compilación, de carga y de ejecución.

<Proxy>: Objeto que representa al objeto después de aplicar el consejo, pueden crearse en tiempo de compilación o en tiempo de ejecución, estáticos o dinámicos

<Introducción>: Mecanismo para añadir nuevos métodos a un objeto (solo con proxies dinámicos)

{Comparación con POO}



{Ventajas y desventajas}

<Lo mejor y lo que puede mejorar del paradigma>

{Ventajas}

- Eliminación de código duplicado, código más limpio
- Mayor enfoque en la lógica principal del código
- Facilita la adaptación del código (Mantenibilidad y Flexibilidad)

}..

{Desventajas}

- Puede añadir mayor costo de desempeño y complejidad
- Dificultad en la depuración
- Fragilidad por dependencia

}..

{Lenguajes de Programación}

<Lenguajes de programación que utilizan POA>

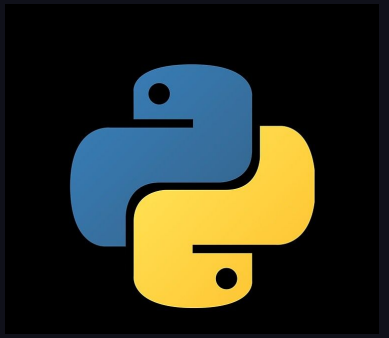
{Lenguajes de Programación POA}

Java

Gracias a AspectJ, Spring AOP, HyperJ, entre otros.

C++

Se cuenta con AspectC++, Grantlee.



Python

Se tiene Spring Python, aunque Python ya soporta POA.

{Lenguajes de Programación POA}

PERL

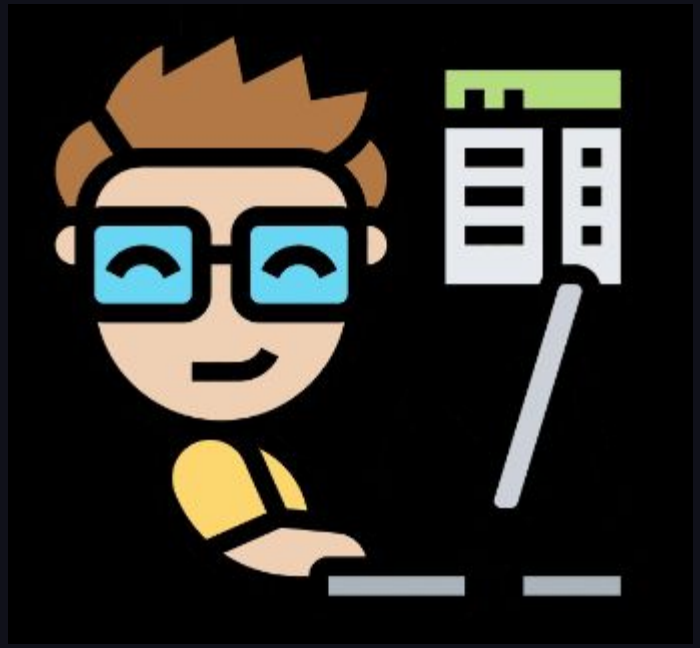
Módulo Aspect PERL permite POA en este lenguaje de programación.

PHP

Existen implementaciones como AOP.io, php-AOP,PHPAspect, entre otras.

Ruby

Gemas como AspectR y aquarium permiten POA en Ruby.



{Ejemplos}

<Ejemplos utilizando Programación Orientada a Aspectos>

{Ejemplos}

<Log de Registro en Python AOP>

```python
def log(message):
    print(message)

def logging_decorator(func):
    def wrapper(*args):
        log(f"Function {func.__name__} called with args: {args}")
        return func(*args)
    return wrapper

@logging_decorator
def add_numbers(*args):
    value = 0
    for n in args:
        value += n
    return value

#Puedes agregar tantos parametros como quieras
print(add_numbers(3, 3, 6))
```

```
Function add_numbers called with args: (3, 3, 6)
12
```

<Python ya soporta AOP debido a los decoradores>

Puedes consultar el código fuente aquí:

Ejemplo Log de Registro

{Ejemplos}

<Seguridad en Python AOP>

```python
def security_decorator(func):
    def wrapper(*args):
        if args[0] in ["Jhonatan", "Miguel", "Paula", "Edgar"]:
          return func(*args)
        else:
          print("Acceso denegado para: " + args[0])
          return None
    return wrapper

@security_decorator
def login(user):
    print("Bienvenido has iniciado sesión como: " + user)

login("Paula")
login("Carlos")
```

```
Bienvenido has iniciado sesión como: Paula
Acceso denegado para: Carlos
```

<¡Más decoradores en Python!>

Puedes consultar el código fuente aquí:

Ejemplo Seguridad

{Ejemplos}

<Sistema de ventas>

```java
@Aspect
public class FinancialTrack {

    ArrayList<Receipt> receipts = new ArrayList<>();
    int balance = 100;

    @Pointcut("execution(* Store.sellProduct(..))")
    public void processOutcomingProducts() {
        // Incoming money
    }

    @Pointcut("execution(* Store.buyProduct(..))")
    public void processIncomingProducts() {
        // Outgoing money
    }

    // Check if we have money for buying products
    @Around("processIncomingProducts()")
    public void processResupply(ProceedingJoinPoint jp) { … }

    // Update balance after selling products
    @AfterReturning(pointcut = "processOutcomingProducts()", returning = "retVal")
    public void processSell(JoinPoint jp, boolean retVal) { … }

}
```

<Ésta vez en Java y AspectJ!>

Puedes consultar el código fuente aquí:

Ejemplo Sistema de Ventas

{Ejemplos}

Reto:

< Modifica el código para que el nuevo usuario, Carlos, sea administrador del sistema >

{Aplicaciones}

<¿Cuándo es mejor usar POA?>

{Aplicaciones}

01 <Logging>

Registro de información en el código, permite hacer depuración y registrar información de ejecución.

02 <Seguridad>

Permite el control de recursos y funciones de aspecto crítico.

03 <Transacciones>

Se definen los aspectos de las transacciones para asegurar operaciones relacionadas, cancelarse en caso de error o garantizar el éxito de la operación.

{Aplicaciones}

04 <Caching>

Implementación que permite el almacenamiento en caché de información para mejorar el rendimiento.

05 <Internacionalización>

Separar contenido y formato de texto disponible en distintos idiomas para su adaptación.

<¡En general POA se utiliza para separar secciones específicas del código!>

{Aplicaciones}

06 <Frameworks de propósito específico>

Introducción de patrones de diseño inspirados en AOP para propósitos particulares, principalmente el desarrollo web.

{Referencias Bibliográficas}

<Material de referencia y consulta>

Bibliografía

- Programación Orientada a Aspectos. http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teoria/Pages/lenguajes
- Pildorasinformaticas. (2021, March 4). Curso Spring. AOP. Vídeo 76. https://www.youtube.com/watch?v=AjXPs9nVHow
- Wikipedia. (2020). Programación orientada a aspectos. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Programación_orientada_a_aspectos
- Tabares B., M. S., Alferez Salinas, G. H., & Alferez Salinas, E. M. (2008). El Desarrollo de Software Orientado a Aspectos: Un Caso Práctico para un Sistema de Ayuda en Línea. Revista Avances en Sistemas e Informática, 5(2), 61-68.

¡Gracias!

< ¿Tienen alguna pregunta? >