

# Programación Orientada a Aspectos

Santiago Cassiano Rozo

Brian Chaparro Cetina

Gustavo Adolfo Mojica Perdigon

Santiago Rodríguez Camargo



# Tabla de contenidos



**01** Filosofía del  
paradigma

**02** Conceptos

**03** Aplicaciones

**04** Ventajas y  
desventajas

---

# Tabla de contenidos



05 Lenguajes

06 Ejemplos

---

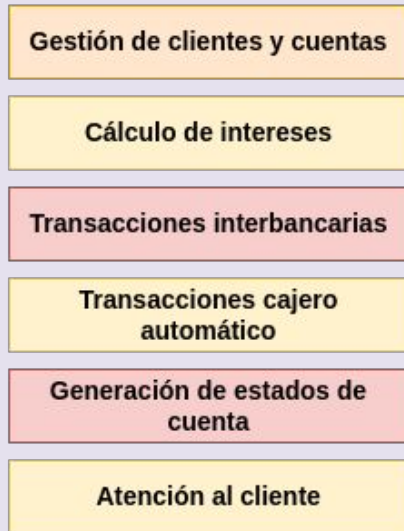


# 01

## Filosofía del paradigma

Comparativa POO con POA

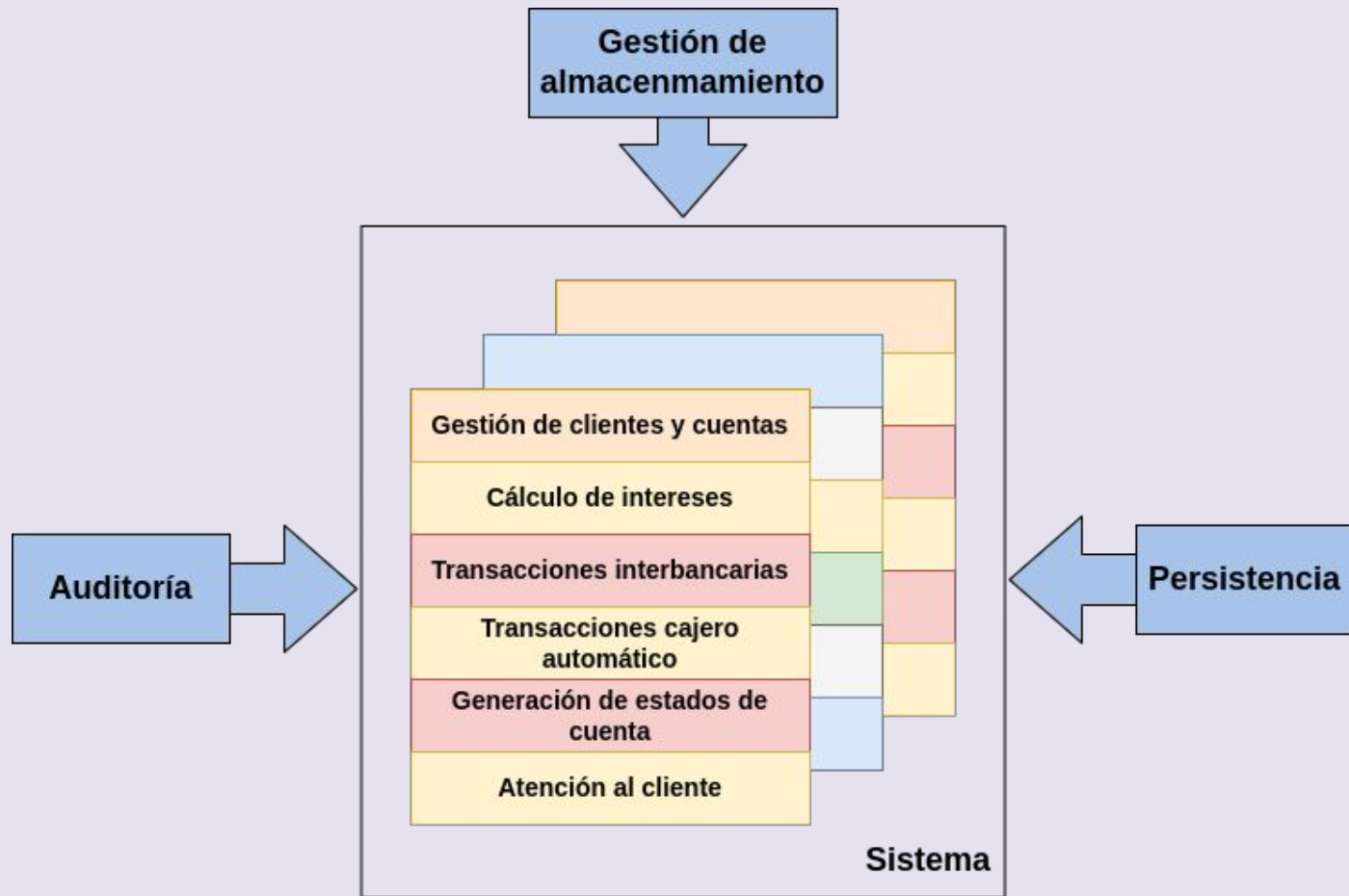




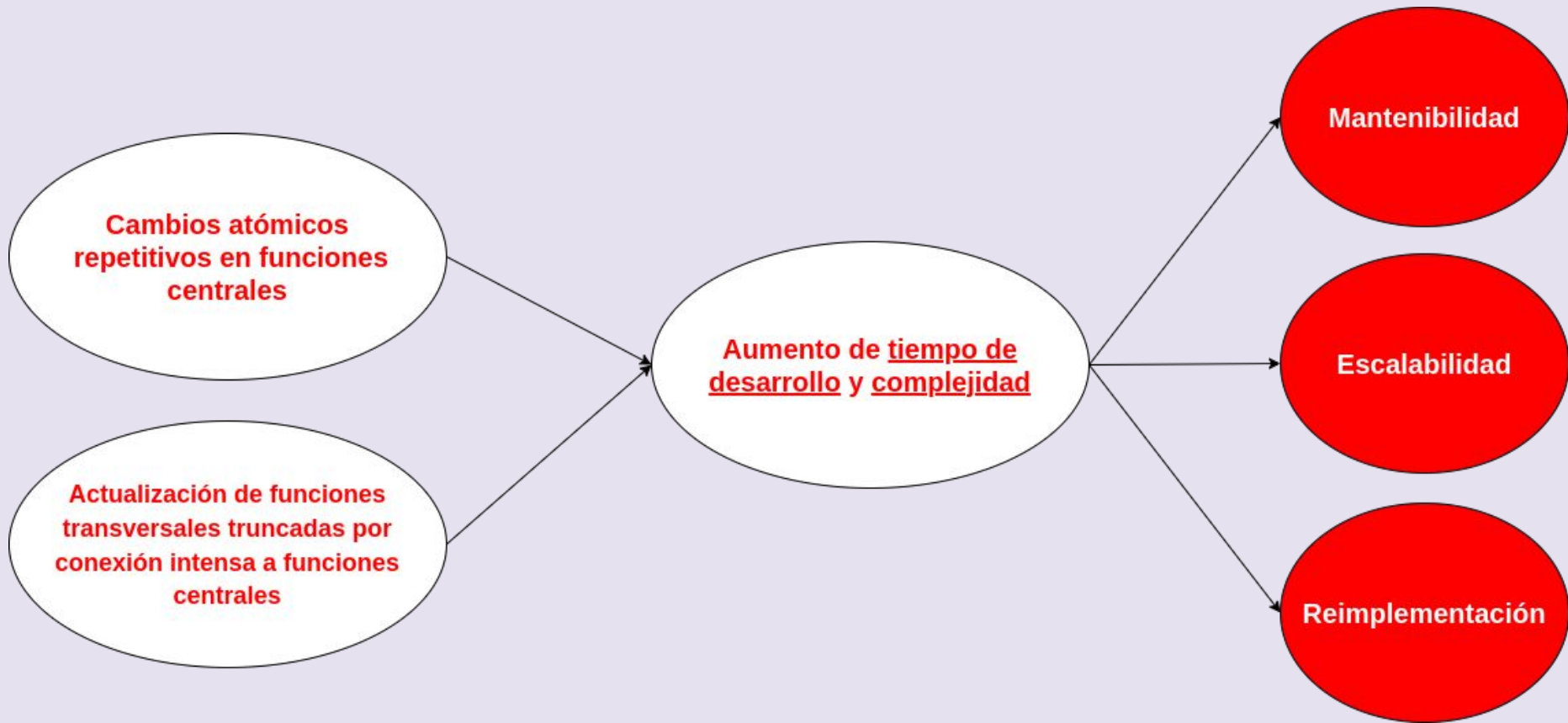
**Funciones Centrales**

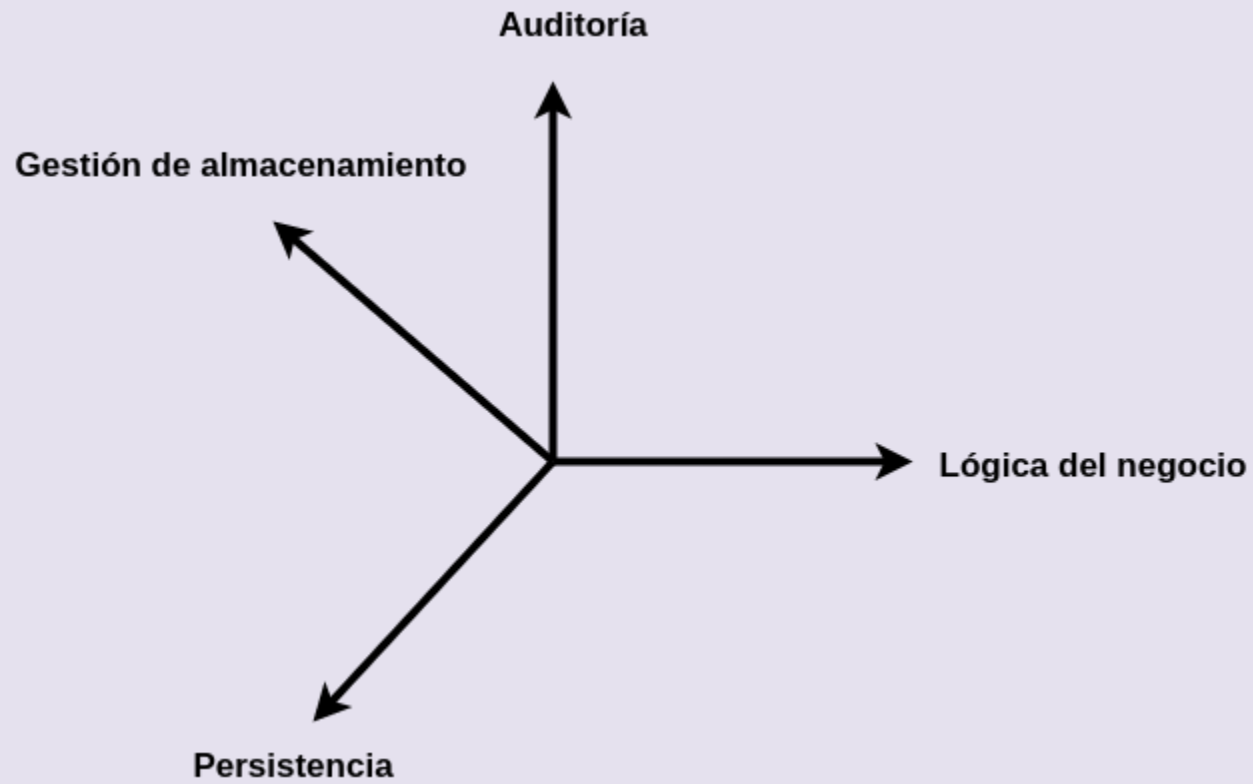


**Funciones Transversales**



Filosofía del paradigma

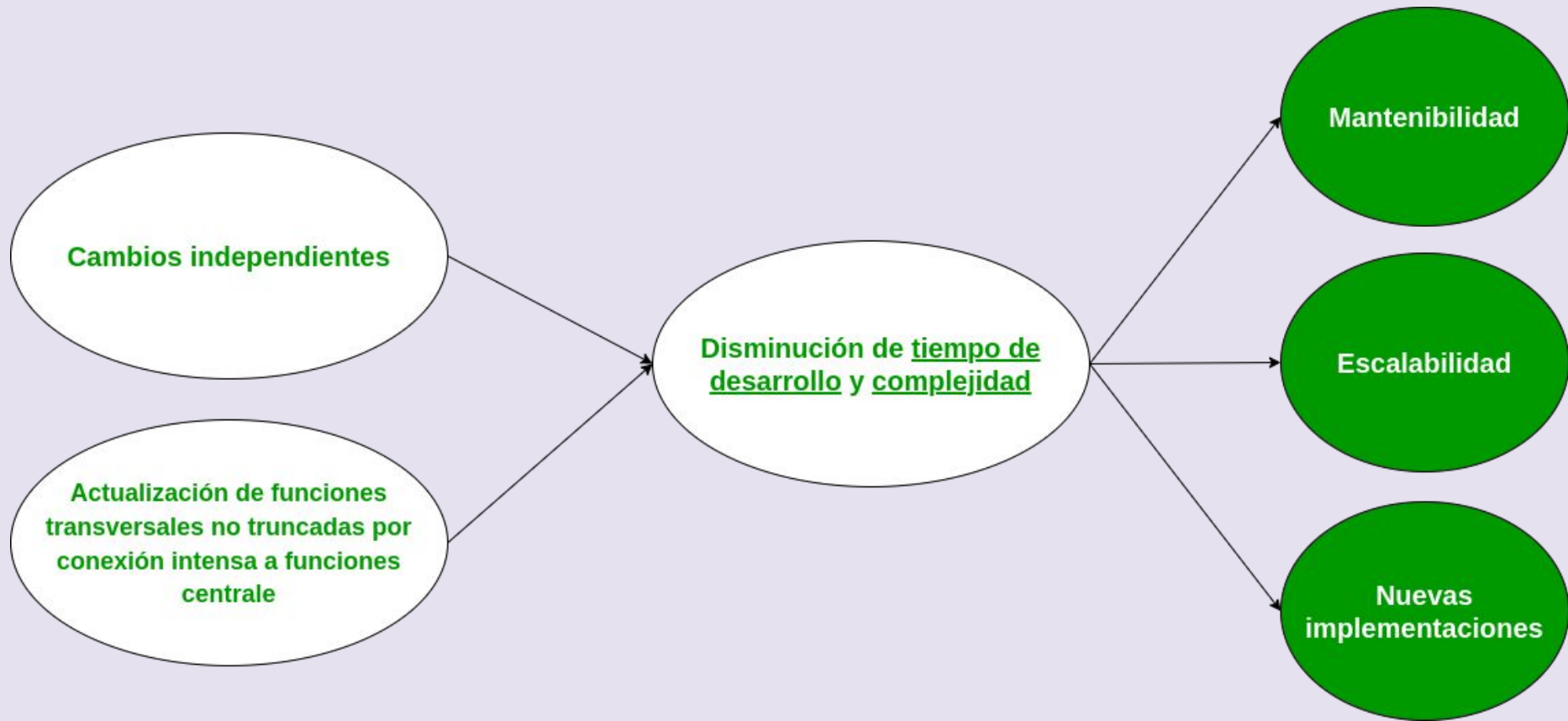




---

**Filosofía del paradigma**





**Funciones transversales = Aspectos**



**"Añadir una nueva modularización (aspectos) para que el desarrollo de las funciones centrales y transversales sea independiente"**

---

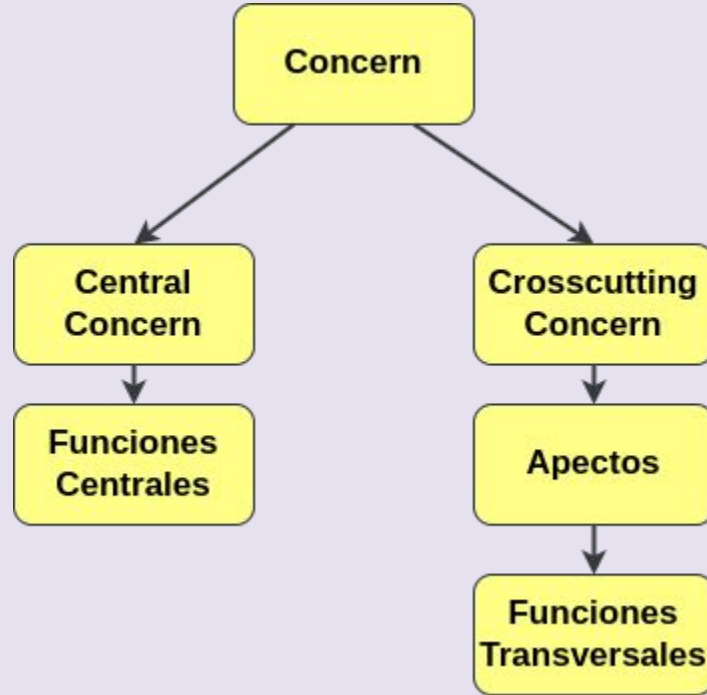
# 02

## Conceptos

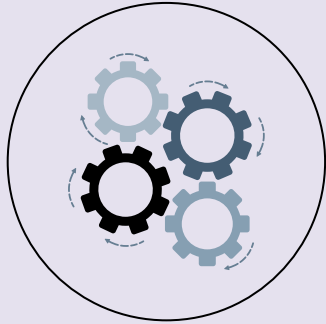
Weaving y conceptos aplicados a AspectJ



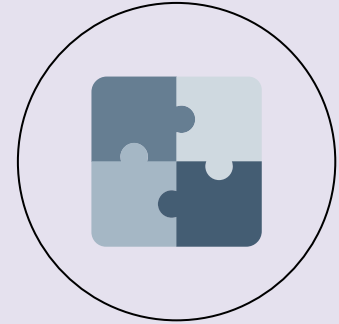
# 1: Concerns



# 2: Weavers



**Entrelazado dinámico  
(Run-time weaver)**



**Entrelazado estático  
(Compile-time weaver)**

# 3: Join point

Cualquier punto de ejecución en un programa:

- Llamada a un método
- Declaraciones y asignaciones
- Retorno en funciones
- Construcción de objetos
- Condicionales
- Comparaciones
- Excepciones
- Ciclos *while* y *do/while*

```
public class Account {  
    void credit(float amount) {  
        _balance += amount;  
    }  
}
```

# 4: Pointcut

Selecciona qué parte en específico del *join point* va a ser elegida:

- Aquí se elige la función de la clase *Account*

A terminal window with a black background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays a pointcut expression in a light blue monospace font: `execution(void Account.credit(float))`.

```
execution(void Account.credit(float))
```

# 5: Advice

Decide qué se hará con el *join point*:

1. *before*
2. *after*
3. *around*
4. *after returning*
5. *after throwing*




# 5: Advice



```
before() : execution(void Account.credit(float)) {  
    System.out.println("About to perform credit operation");  
}
```

# 6: Introduction

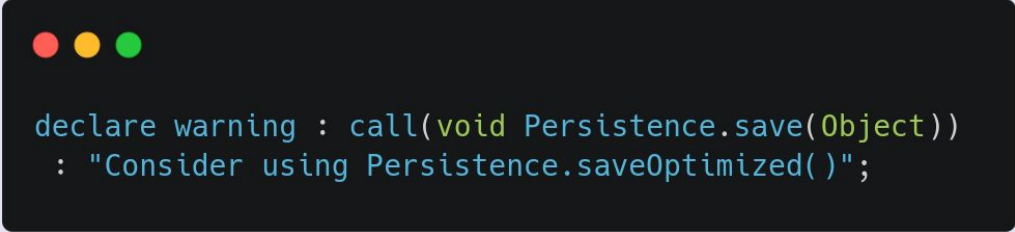
Añade instrucciones a las *clases*, *interfaces* y *aspectos* sin afectarlos directamente:



```
declare parents: Account implements Persistence;
```

# 7: Compile-time-declaration

Añade *warnings* y *errores* sin afectar directamente las *clases*, *interfaces* y *aspectos*:



```
declare warning : call(void Persistence.save(Object))
: "Consider using Persistence.saveOptimized()";
```

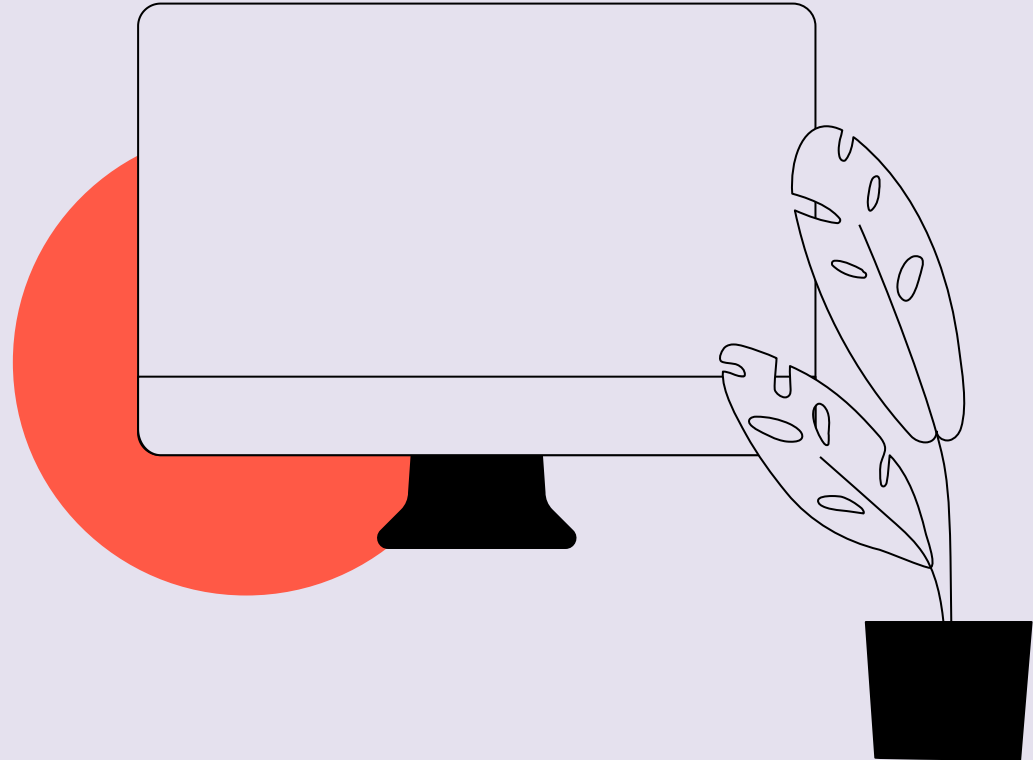
# 8: Aspecto (continuación)

```
public aspect ExampleAspect {  
  
    //pointcut  
    execution(void Account.credit(float))  
  
    //advice  
    before() : execution(void Account.credit(float)) {  
        System.out.println("About to perform credit operation");  
    }  
  
    //introduction  
    declare parents: Account implements Persistence;  
  
    //compile-time declaration  
    declare warning : call(void Persistence.save(Object))  
        : "Consider using Persistence.saveOptimized()";  
}
```

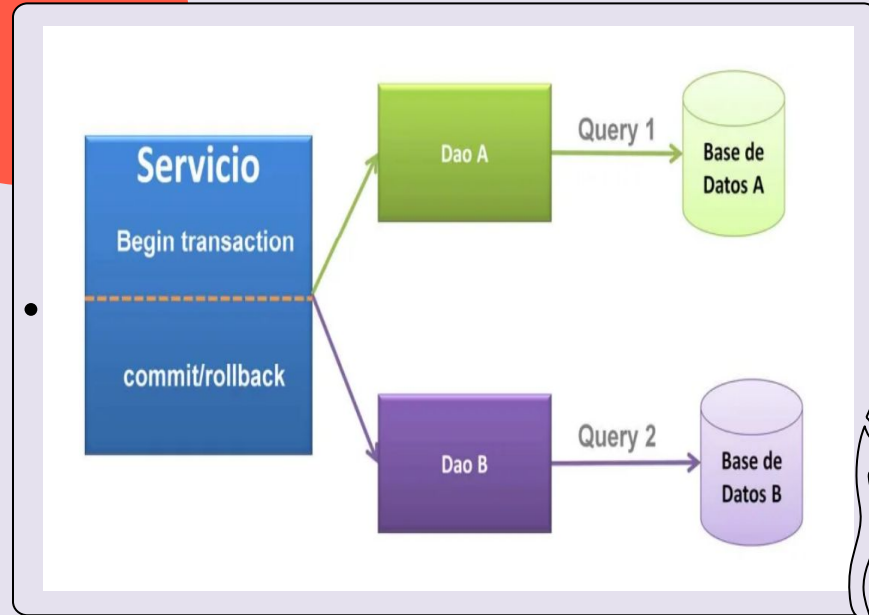
---

# 03

## Aplicaciones



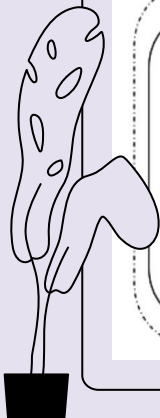
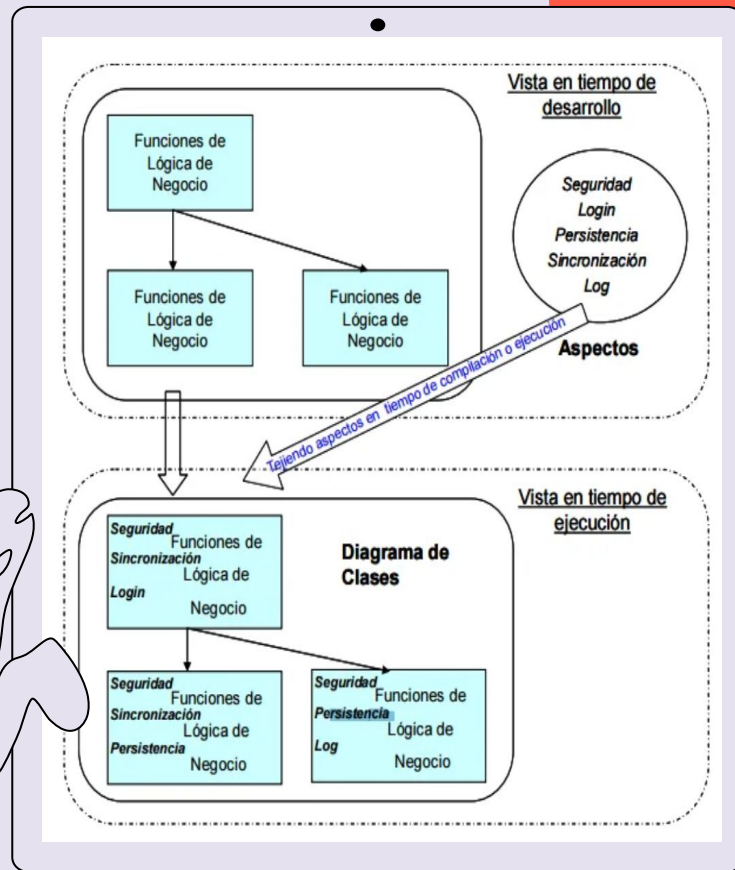
# Manejo de transacciones



En la figura podemos observar un método de servicio que ejecuta llamadas a más de un DAO, y a su vez cada DAO modifica el estado de la base de datos al escribir y/o modificar su información.

# Sincronización

La sincronización expuesta aquí se basa en los procesos, otorga varias técnicas de coordinación de hilos que se ejecutan en forma simultánea.



---

# Manejo de Excepciones

## Soporte

Al crear un mecanismo de gestión y manejo de excepciones, no deberíamos partir desde cero, ya que existe un montón de excelentes productos y frameworks.

## Implementación

El primer paso es crear el atributo (Attribute) que decorará las clases/ interfaces para capturar su ejecución e inyectar el código de manejo de excepciones.





---

# Otras aplicaciones...

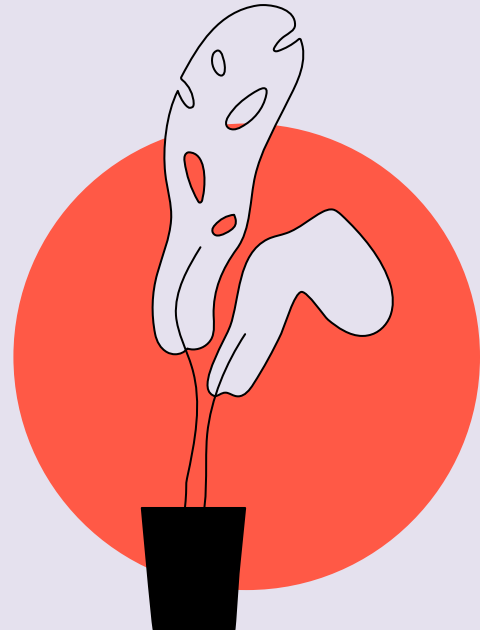
## Control de Acceso

Debido al uso eficiente del paradigma dentro del uso de las cuentas y perfiles, también afecta directamente el uso de los controles de acceso y la seguridad.



Para realizar la extensión UML 2.0, la propuesta de extensión del perfil, no permite modificar al metamodelo existente, sino adaptarlo.

## Perfiles





04

## Ventajas y desventajas

---

# Ventajas

<b>Modularización</b>	Menor dependencia
<b>Menos redundancia</b>	Código más limpio
<b>Agilización y organización</b>	Trabajo en equipo
<b>Adaptabilidad</b>	POO +
<b>Test de calidad</b>	Speed

```
namespace Examples\Forum\Domain\Model;
use Examples\Forum\Logger\ApplicationLoggerInterface;

class Forum {

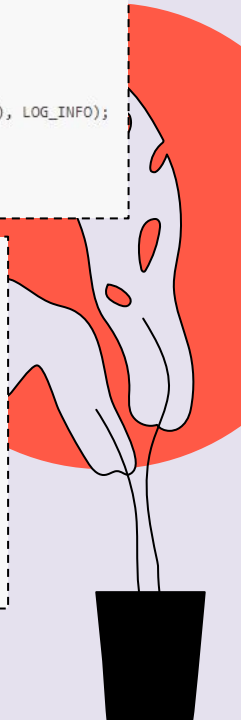
    /**
     * @Flow\Inject
     * @var ApplicationLoggerInterface
     */
    protected $applicationLogger;

    /**
     * Delete a forum post and log operation
     */
    public function deletePost(Post $post): void
    {
        $this->applicationLogger->log('Removing post ' . $post->getTitle(), LOG_INFO);
        $this->posts->remove($post);
    }
}
```

```
namespace Examples\Forum\Domain\Model;

class Forum {

    /**
     * Delete a forum post
     */
    public function deletePost(Post $post): void
    {
        $this->posts->remove($post);
    }
}
```



# Desventajas



<b>Antipatrón</b>	Acciones inesperadas
<b>Debugging</b>	Flujo no explícito
<b>Cuidado</b>	Cambios y errores
<b>Sobrecarga</b>	Weaving
<b>Críticas</b>	Fragilidad

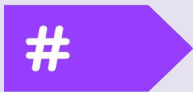


---

# 05

## Lenguajes





**C# (.Net)**

Extensión  
postsharp



**C/C++**

Extensión  
AspectC++



**Java**

AspectJ



**Python**

Decorators  
Aspyct



**JavaScript**

Decorators  
KAOP





# 06

## Ejemplos

---

# Registro de actividades - Java - Spring



**Una tienda que compra y vende productos quiere mantener un registro de cuando realiza sus operaciones.**



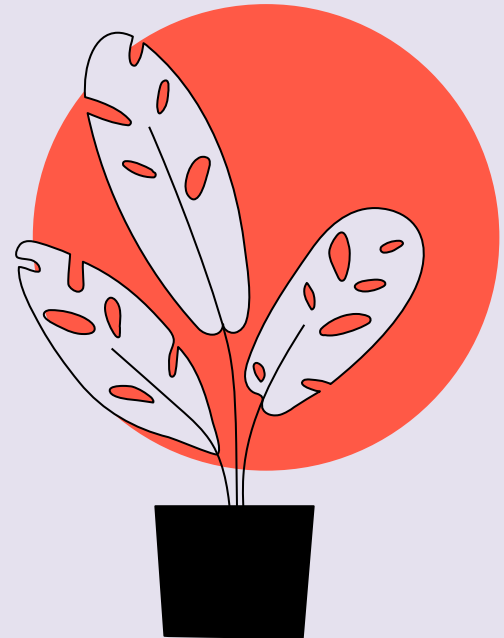
---

# Control de acceso - Python - Decoradores

Utilizando decoradores se buscará verificar que antes de que un usuario ejecute una función, dicha persona si tenga permisos para realizarla.

Codigo del ejemplo:

[https://colab.research.google.com/drive/1H8R0y4DDeC-AjrrbivMDEsWM3OM0Oyk\\_?usp=sharing](https://colab.research.google.com/drive/1H8R0y4DDeC-AjrrbivMDEsWM3OM0Oyk_?usp=sharing)



# Referencias

- León, P. (2021, December 7). Una aproximación a la Programación Orientada a Aspectos. Medium.  
<https://medium.com/@PabloLeonPsi/una-aproximaci%C3%B3n-a-la-programaci%C3%B3n-orientada-a-aspectos-a62d377ebe79>
  - Laur Spilca. (2019, May 24). Spring Framework - Lesson 4 - Aspect Oriented Programming - AOP [Video]. YouTube. <https://www.youtube.com/watch?v=BVk54NRRFsY>
  - pildorasinformaticas. (2021, March 4). Curso Spring. AOP. Vídeo 76 [Video]. YouTube. <https://www.youtube.com/watch?v=AjXPs9nVHow>
  - Programación Orientada a Aspectos.  
[http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa\\_teor%C3%ADa/Pages/lenguajes.html](http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teor%C3%ADa/Pages/lenguajes.html)
  - Aspect-oriented programming. Flow Framework. Retrieved November 20, 2022, from <https://flowframework.readthedocs.io/en/stable/TheDefinitiveGuide/PartIII/AspectOrientedProgramming.html>
-

# Referencias

- miw-upm. (2016). Programación Orientada a Aspectos con Spring. YouTube. Retrieved November 20, 2022, from <https://www.youtube.com/watch?v=oFm4qtfyUh4&t=10s>.
  - Decoradores en Python.. CódigoFacilito. <https://codigofacilito.com/articulos/decoradores-python>
  - Javier Fernandes. Programación Orientada a Objetos. Programación Avanzada con Objetos Disponible en <https://sites.google.com/site/programacionhm/conceptos/aop>
  - Laddad, R. (2003) Aspectj in action: Practical Aspect-Oriented Programming. Greenwich, CT: Manning.
-



# Gracias

---