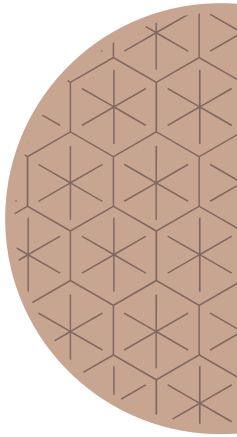
Programación orientada a aspectos

Ricardo Andrés Calvo Méndez
Javier Alejandro Ortiz Silva
Emmanuel Steven Rojas Arcila
Santiago Vargas Avendaño

Lenguajes de programación 2021 I
Universidad Nacional de Colombia

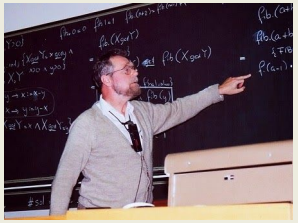
Temas

1. Filosofía del paradigma
2. Conceptos claves
3. Ventajas y desventajas
4. Lenguajes de programación
5. Aplicaciones de paradigma
6. Diagramas UML
7. Proceso de compilación
8. Ejemplos en distintos lenguajes (Java)
9. Ejemplos en distintos lenguajes (AspectJ)



Filosofía

Separación de conceptos o intereses - Edsger Dijkstra (1974)



Es un principio de diseño que busca **separar un programa informático en secciones distintas** de forma tal que cada que cada sección se enfoque un interés delimitado.

→ Modularidad y encapsulamiento

Ley Demeter - Ian Holland (1987)

Es un principio de diseño que establece que **cada unidad debe tener solamente un conocimiento limitado sobre las otras unidades** y solo únicamente aquellas unidades estrechamente relacionadas.

→ Encapsulamiento y bajo acoplamiento

Historia

A inicios de los noventa se investiga fuertemente el área de *SoC*

Algunas técnicas para lograr *SoC:*

- **Programacion meta-nivel**
  → Programas operan sobre programas
- **Filtros de descomposición**
  → Instancias de clase filter gestionan mensajes
- **Programacion adaptativa**
  → uso de patrones de código

El grupo Demeter del Xerox Palo Alto Research center (PARC) estudia a nivel teórico y de implementación este campo.

En 1995 se adopta el nombre *Aspect oriented programming* que se usará en las publicaciones del grupo.

Gregor Kickzales

Karl Lieberherr

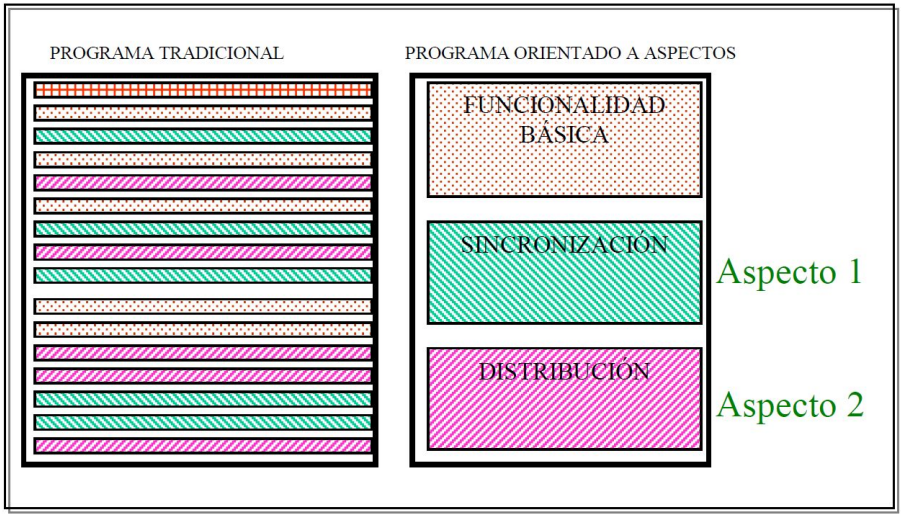
Cristina Lopes

Mira Mezini

Idea central de la POA

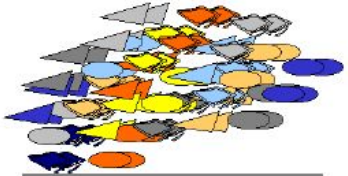
Las obligaciones o conceptos transversales son tratadas como módulos separados.

- Una obligación transversal ***se repite en varias partes*** de un programa sin importar si las secciones en las que aparece tienen relación.
- Se separan ***componentes*** de ***aspectos*** creando un mecanismo para abstraerlos y unirlos para armar todo el sistema.



Evolución de los sistemas de software

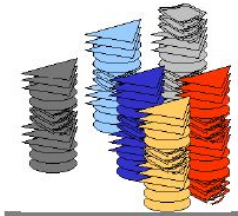
Saber descomponer sistemas complejos en partes más fáciles de manejar



1ª Generación:
Código espaghetti

No hay separación entre datos y funcionalidad.

- Se dificulta la modificación y el mantenimiento del sistema



2ª y 3ª Generación:
Descomposición funcional

El sistema se organiza según las funciones definidas en el dominio del problema.

+ Resulta fácil integrar nuevas funciones.
- Difícil gestión de los datos compartidos entre funciones

4ª Generación:
Descomposición en objetos

Se usa la abstracción para modelar el problema en sus objetos

+ Resulta fácil integrar nuevos datos
- Crear algunas funciones implica modificar varias clases

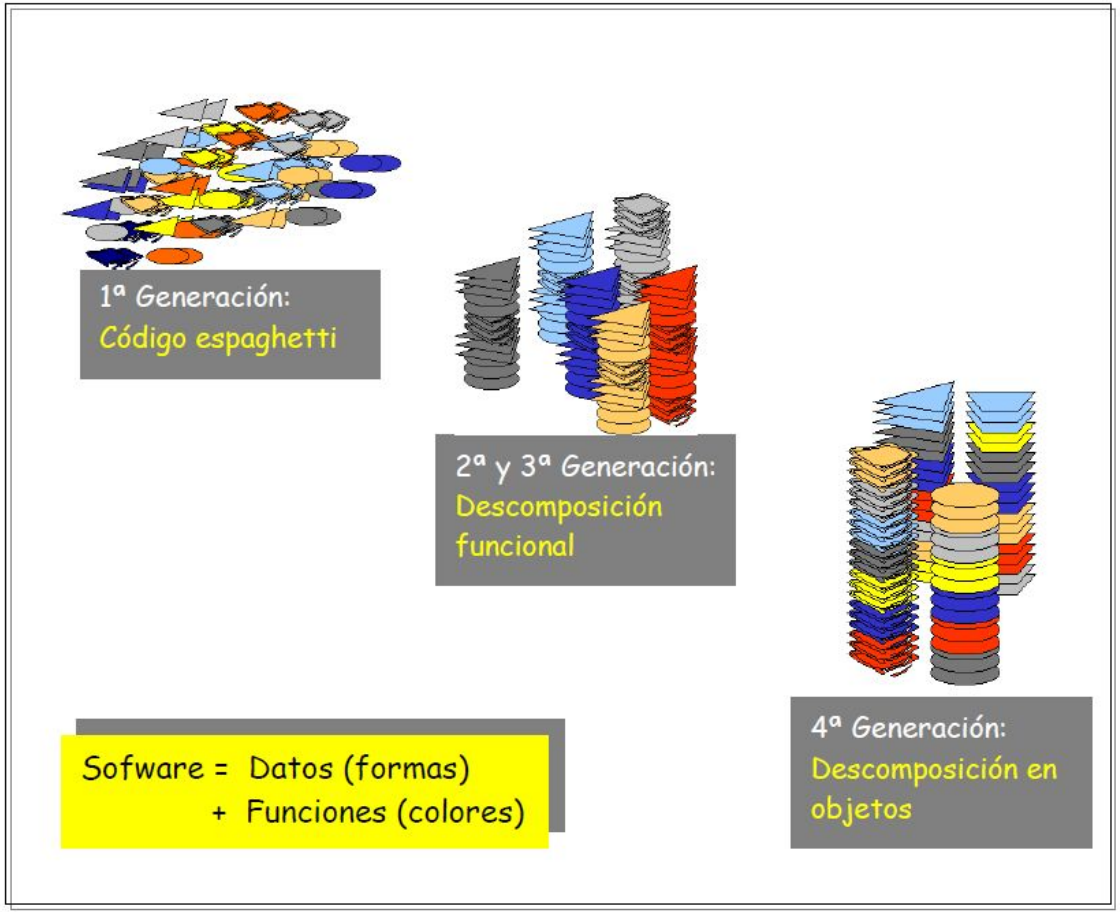


Figura 1 Esquema de la evolución de la ingeniería del software

Con el paradigma de POA es posible generar un modelo adaptado a la realidad y útil para la solución del problema

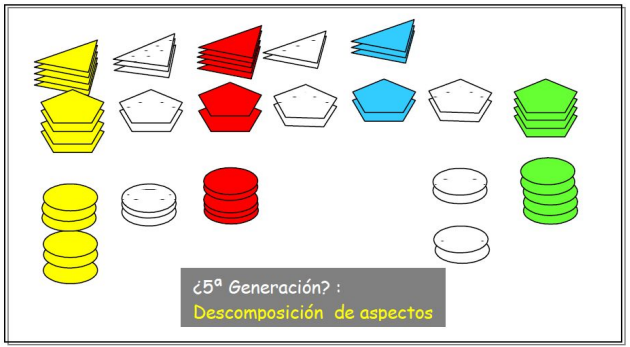


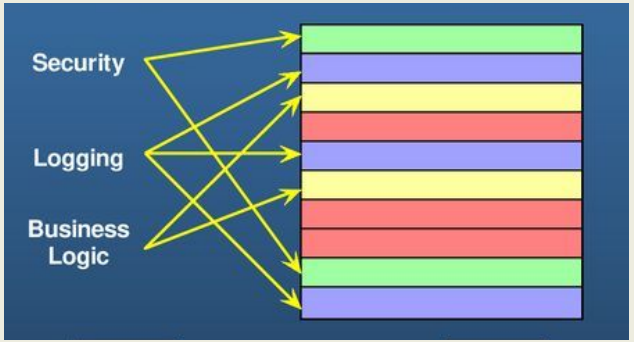
Figura 2. Descomposición en aspectos ¿Una quinta generación?

Problemas de POO

Usando POO o otras técnicas de abstracción de alto nivel, se logra un diseño y una implementación funcional. Sin embargo, existen conceptos que no pueden encapsularse dentro de una unidad funcional, debido a que atraviesan todo el sistema o varias parte de él (crosscutting concerns).

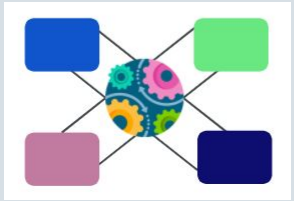
1. Código Mezclado (Code Tangling)

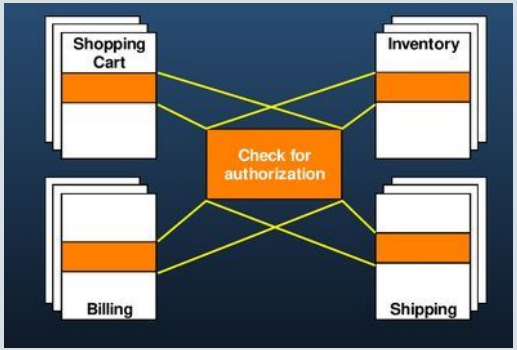
En un mismo módulo de un sistema de software pueden simultáneamente convivir más de un requerimiento. Esta ***múltiple existencia de requerimientos*** lleva a la presencia conjunta de elementos de implementación de más de un requerimiento.



2. Código Diseminado (Code Scattering)

Como los requerimientos están esparcidos sobre varios módulos, la ***implementación*** resultante también queda ***diseminada*** sobre esos módulos.



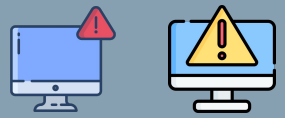


Consecuencias la mala gestión de conceptos transversales

- Baja correspondencia
- Menor reuso
- Baja calidad de código
- Menor productividad
- Difícil evolución

- Sincronización

- Gestión de errores y fallas



- Manejo de memoria

- Seguridad

- Redes

Conceptos

Concepto (Concern)

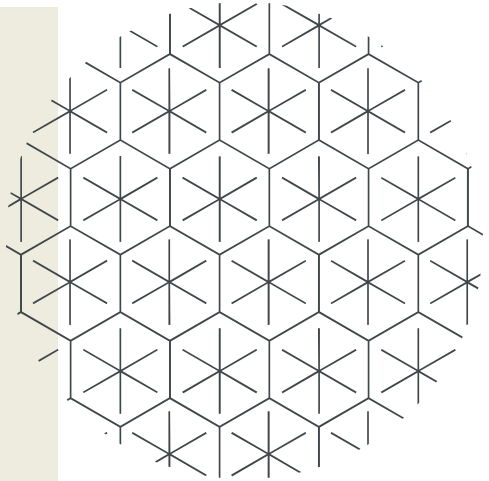
- Requerimiento que se debe implementar para satisfacer las necesidades del sistema.

Componente (Component)

- Partes de código que pueden ser encapsuladas.

Aspecto (Aspect)

“Un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño.” (G. Kiczales).

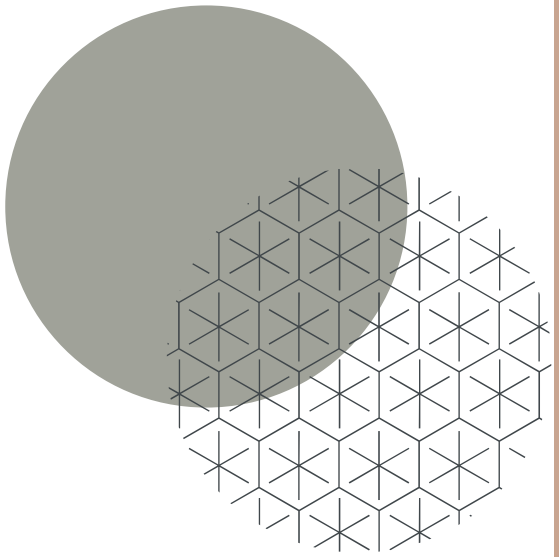


Separación de conceptos (separation of concerns)

encapsular, abstraer y separar los componentes de los aspectos.

usualmente se utilizan lenguajes diferentes para programar los aspectos y los componentes.

Por lo general se usan lenguajes orientados a objetos como Java para implementar los componentes.



Punto de unión/enlace (Joint point)

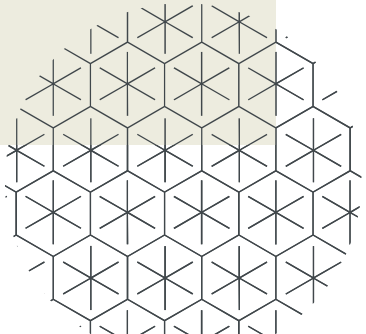
Punto donde un aspecto puede ser conectado:

- Llamada a un método.
- Lanzamiento de una excepción.
- Modificación de un campo.

Consejo (Advice)

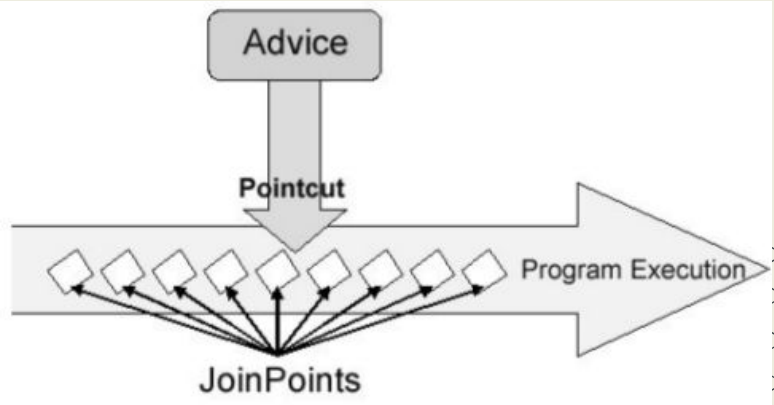
Es la implementación del aspecto, y se insertan en la aplicación en los Puntos de unión.

- before
- after returning
- after throwing
- after
- around



Puntos de corte (Pointcut)

Define los Consejos que se aplicarán a cada Punto de Cruce. Un pointcut es un predicado o condición para la aplicación de un aspecto.



Introducción (Introduction)

Permite añadir métodos o atributos a clases ya existentes

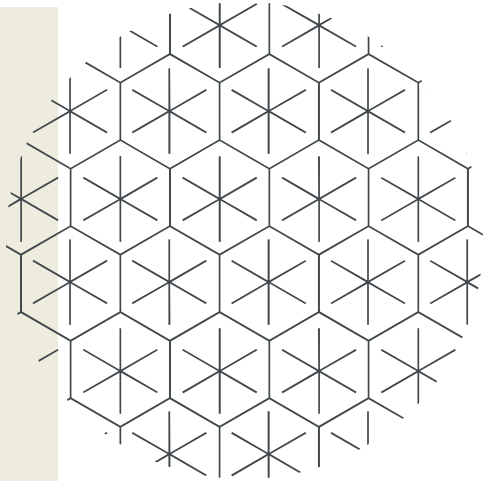
Destinatario (Target)

Es la clase aconsejada, la clase que es objeto de un consejo.

Resultante (Proxy)

Es el objeto creado después de aplicar el Consejo al Objeto Destinatario.

Tejedor (Weaver)



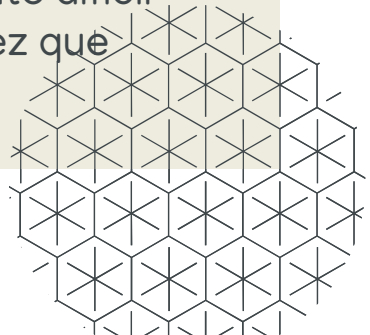
Se encarga de mezclar los diferentes componentes con los aspectos, ayudándose de las reglas de recomposición que proporcionan los puntos de enlace

El resultado de este proceso de tejido (o weaving) es un código ejecutable de todo el sistema

Entrelazado Estático

El entrelazado estático implica modificar el código fuente de una clase insertando sentencias en estos puntos de enlace.

Evita que el nivel de abstracción que se introduce con la programación orientada a aspectos se derive en un impacto negativo en el rendimiento de la aplicación. Pero, por el contrario, es bastante difícil identificar los aspectos en el código una vez que éste ya se ha tejido.



Yuxtaposición

Intercalación del código de los aspectos con los componentes.

Mezcla

Todo el código queda mezclado con una combinación de descripciones de componentes y aspectos

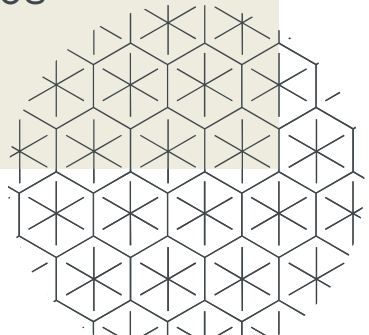
Fusión

Los puntos de enlace no se tratan de manera independiente, se fusionan varios niveles de componentes y de descripciones de aspectos en una acción simple.

Entrelazado Dinámico

Los aspectos como las estructuras entrelazadas se deben modelar como objetos y deben mantenerse en el ejecutable.

El tejedor es capaz de añadir, adaptar y borrar aspectos de forma dinámica, si así se desea, durante la ejecución. El tejedor también tiene en cuenta el orden en el que se entremezclan los aspectos.



Ventajas
Y
desventajas

Ventajas

- Ayuda a superar los problemas causados por: **Código Mezclado** y **Código Diseminado.**

- **Implementación modularizada:** POA logra separar cada concepto con mínimo acoplamiento. Llevando a un código más limpio, menos duplicado, más fácil de entender y de mantener.

- **Mayor evolucionabilidad:** La separación de conceptos permite agregar nuevos aspectos, modificar y / o remover aspectos existentes fácilmente.

Ventajas

- **Capacidad de retrasar las decisiones de diseño:** Permite implementar requerimientos separadamente, e incluirlos automáticamente en el sistema.
- **Resuelve el dilema del arquitecto:** ¿Cuántos recursos invertir en el diseño? ¿Cuándo es “demasiado diseño”?
- **Mayor reusabilidad:** Mayor probabilidades de ser reusados en otros sistemas con requerimientos similares.

Ventajas

- **Divide y vencerás:** se separan la funcionalidad básica de los aspectos.
- **N-dimensiones:** Se tiene la posibilidad de implementar el sistema con las dimensiones que sean necesarias.
- **Mínimo acoplamiento y máxima cohesión:** Se realizan cada una de sus partes por separado y se unen, estas partes, en este caso los aspectos interactúen adecuadamente.

Desventajas

- Posibles choques entre **el código funcional** y **el código de aspectos.**
- **Posibles choques entre los aspectos.**
- **Problemas propios del desarrollo:** surgen problemas a medida que se desarrolla. Las tecnologías que implementan POA deben estar en constante actualización y cambio, lo cual genera problemas de compatibilidad con versiones diferentes.

Desventajas

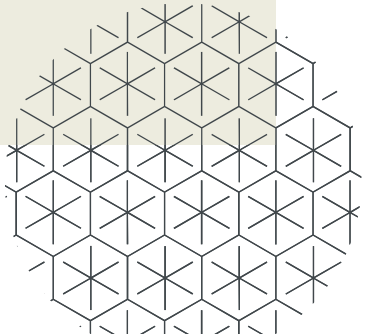
- Posibles choques entre **el código de aspectos** y **los mecanismos del lenguaje.**

- **Documentación:** No cuenta con una documentación robusta al respecto. Por lo que puede suponer problemas para desarrolladores con baja experiencia o para proyectos que dependan en gran parte de este paradigma.

Lenguajes de programación

Son aquellos lenguajes que permiten separar la definición de la funcionalidad “principal” de la definición de los diferentes aspectos.

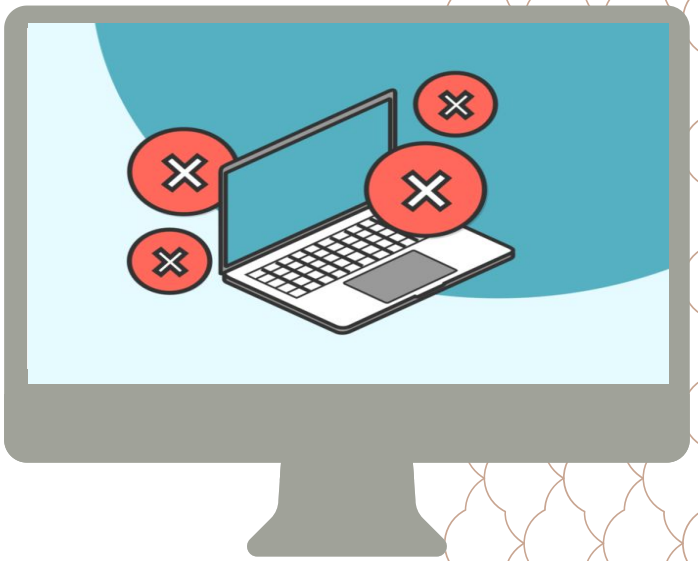
- Debe ser claramente identificable.
- Debe auto contenerse.
- Debe ser ser fácilmente modificable.
- No deben interferir entre ellos



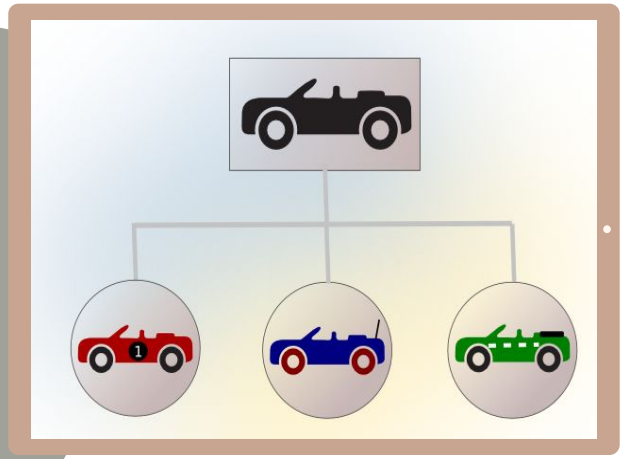
LOA de dominio específico

Han sido diseñados para soportar algún tipo particular de aspectos, como por ejemplo la concurrencia, sincronización o distribución.

Algunos de estos lenguajes necesitan imponer restricciones en el lenguaje base, para garantizar que las incumbencias que son tratadas en los aspectos no puedan ser programadas en los componentes.

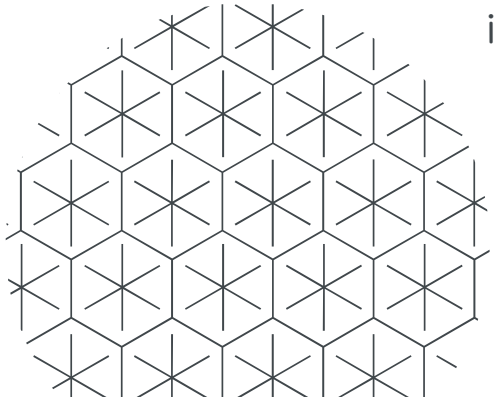


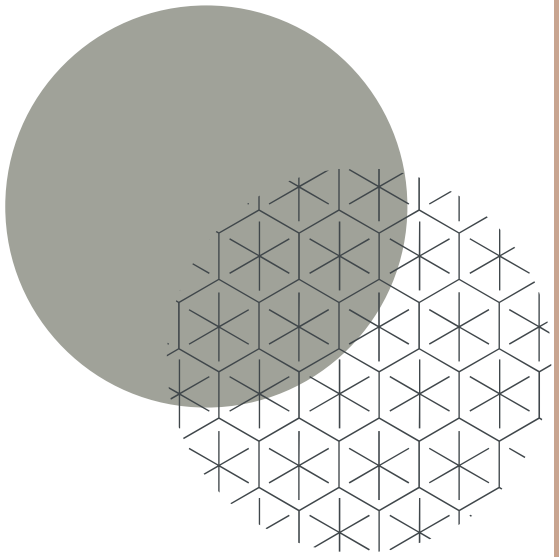
LOA de propósitos generales



Han sido diseñados para soportar cualquier tipo de aspectos.

Este tipo de lenguajes no pueden imponer restricciones en el lenguaje base. Generalmente tienen el mismo nivel de abstracción que el lenguaje base, y soportan las mismas instrucciones o primitivas del lenguaje base.





Problema de los lenguajes base

Hay dos alternativas relativas al lenguaje base:

- Diseñar un nuevo lenguaje base junto con el lenguaje de aspectos.
- Tomar un lenguaje ya existente como base, ya que la base ha de ser un lenguaje de propósito general.

JPAL

Enfatiza los puntos de enlace, ya que son especificados independientemente del lenguaje base

COOL

Son comprendidos por un conjunto de módulos coordinadores.

RIDL

Es un LADE que maneja la transferencia de datos entre diferentes espacios de ejecución.

AspectJ

Extensión Java del proyecto Eclipse para soportar el manejo de aspectos

Python

No necesita de ninguna librería para hacer POA, esto se logra mediante los decorators.

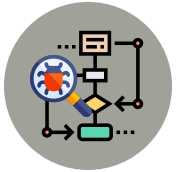
Spring AOP

Es implementado en java puro con las anotaciones de java @Aspect o basado en esquema con un XML

Aplicaciones: Casos de uso típicos

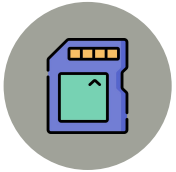
Seguridad y perfilamiento

- Datos confidenciales
- Control de acceso



Tracing

- Entender el flujo del programa
- Solución de problemas



Manejo de la Memoria

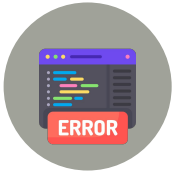
- Se evita repeticion
- Procesos más eficientes

Gestión de transacciones

- Acceso y uso transversal de datos
- Spring AO y Hibernate ORM

Monitoreo del rendimiento

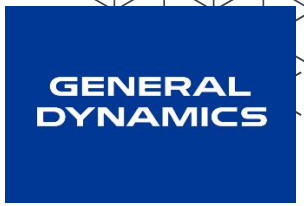
- Point cuts-Join points-Advices
- Generacion de estadisticas



Manejo de errores

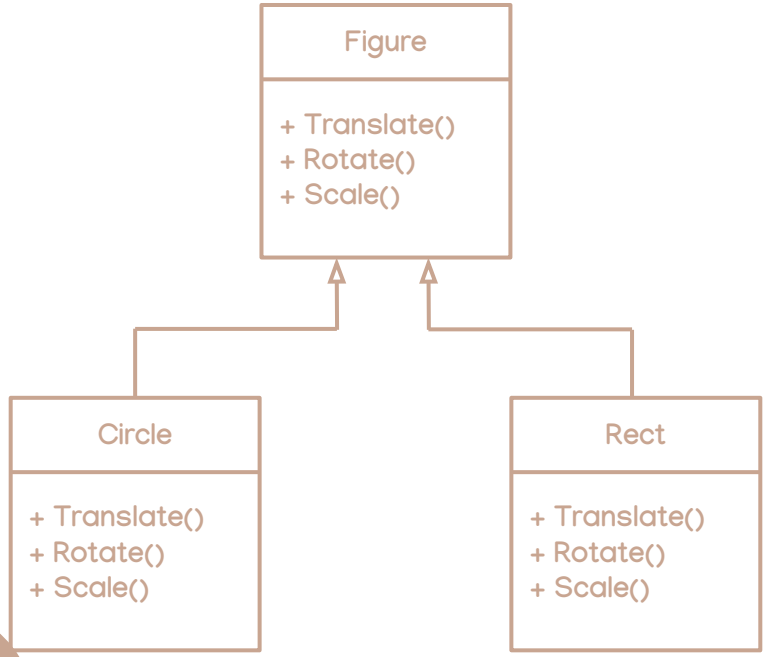
- Política clara para cada excepción

Aplicaciones: Empresas que han trabajado AOP



Aplicaciones: Proyectos que emplean AOP

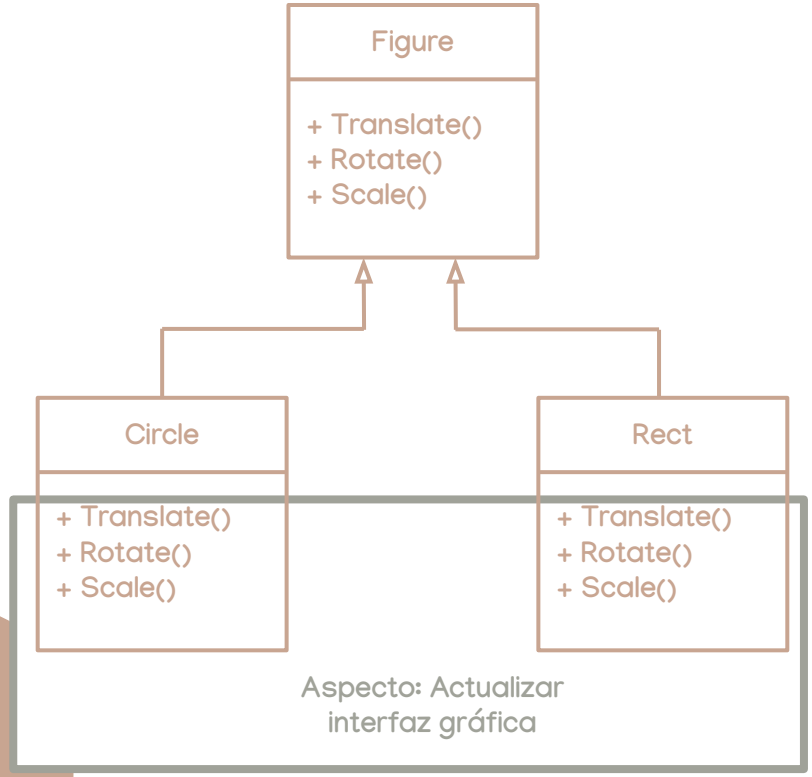
Diagrama UML



Los aspectos no tienen una representación en los diagramas de clases UML.

Sin embargo, se puede hacer una aproximación a su representación para así mostrar gráficamente donde se encontrarían en estos diagramas.

Diagrama UML



En este ejemplo encontramos la clase Figura con tres métodos: Trasladar, Rotar y Escalar.

Existen dos clases hijas Circulo y Rectangulo que sobreescriben los métodos de Figura.

Los métodos ya sobreescritos deben actualizar la interfaz gráfica, por lo tanto todos estos son cubiertos por un aspecto como se muestra en la representación.

Proceso de compilación

- Compiladores

Javac

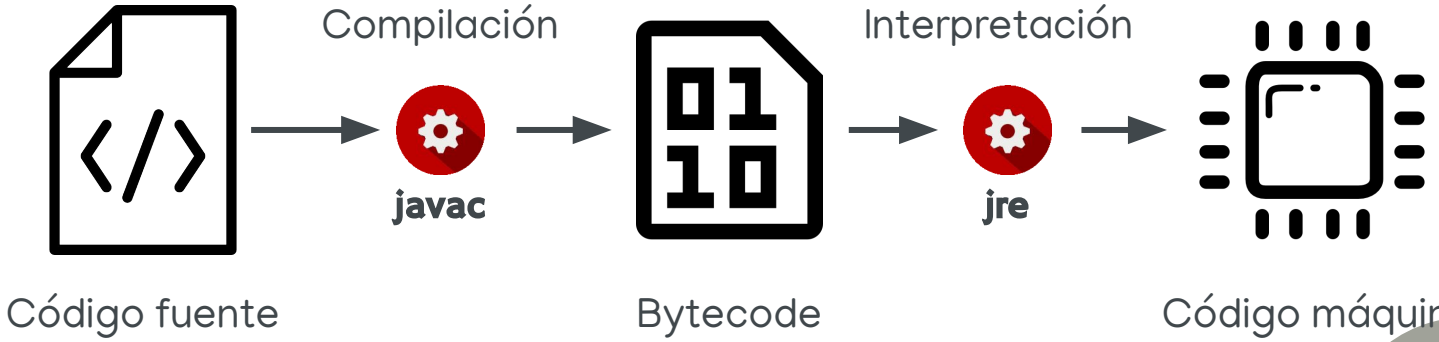
Java compiler

AJC

AspectJ compiler

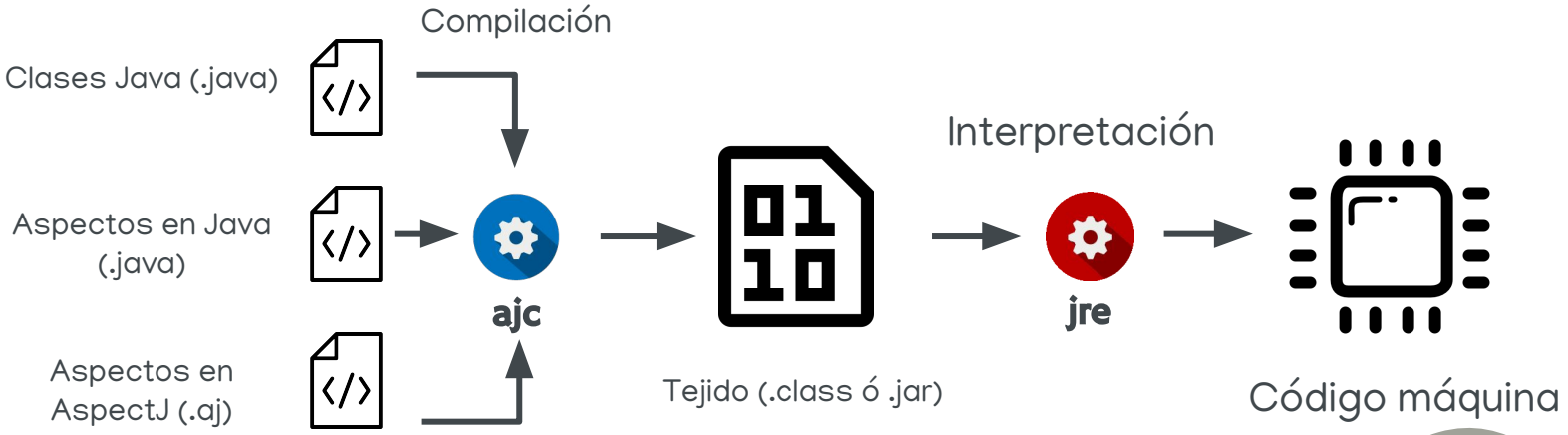
Proceso de compilación

- Compilación de Java



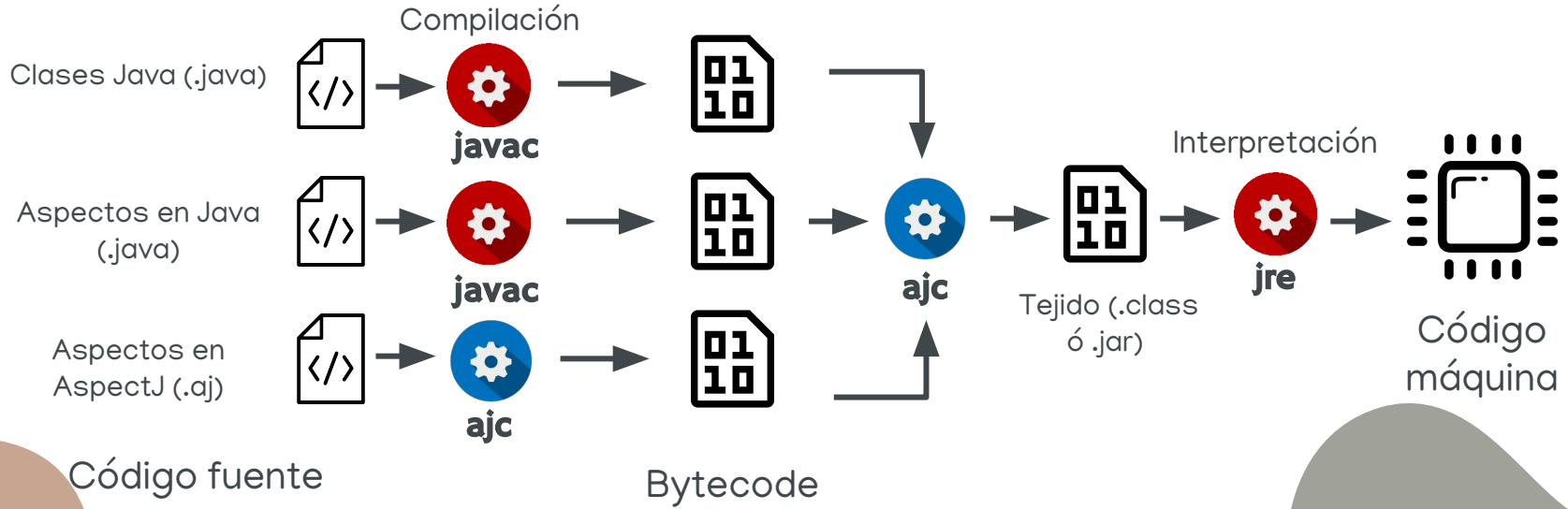
Proceso de compilación

- Compilación de Java + AspectJ <AspectJ>



Proceso de compilación

- Compilación de Java + AspectJ <AspectJ>



Ejemplos

IntelliJ Idea

- Hello World
- Cuenta bancaria

Referencias

- Libro AspectJ in action, segunda edición: https://livebook.manning.com/book/aspectj-in-action-second-edition/chapter-8/45
- Configuración de AspectJ en IntelliJ IDEA: https://www.jetbrains.com/help/idea/aspectj.html
- http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teoria/index.html
- Visión General de la Programación Orientada a Aspectos: https://www.researchgate.net/profile/Antonia-Reina-Quintero/publication/253410957_Vision_General_de_la_Programacion_Orientada_a_Aspectos/links/0f3175340507d3f5ac000000/Vision-General-de-la-Programacion-Orientada-a-Aspectos.pdf
- Aspect-Oriented Programming: An Historical Perspective(What's in a Name?): http://isr.uci.edu/tech_reports/abstracts/UCI-ISR-02-5-abs.pdf
- PROGRAMACIÓN ORIENTADA A ASPECTOS Análisis del paradigma https://www.angelfire.com/ri2/aspectos/Tesis/tesis.pdf

Muchas gracias