

Aspectos - Servicios



POO

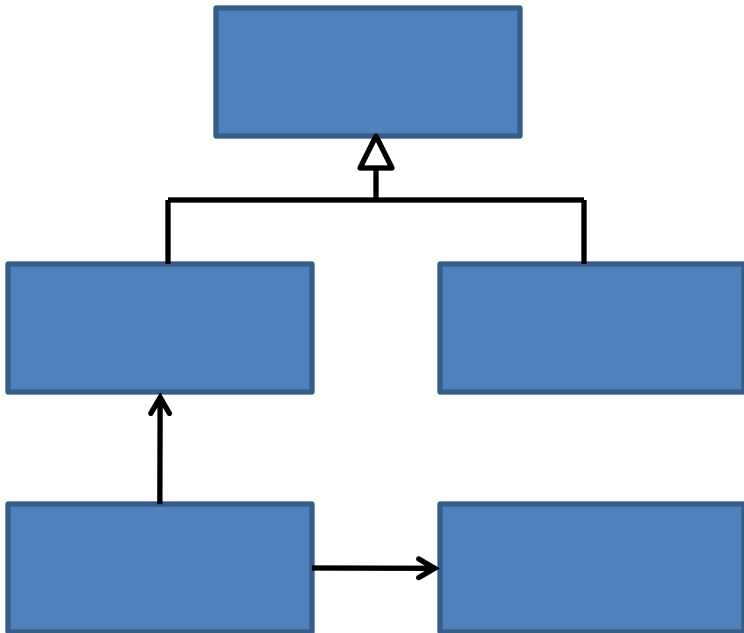
- Manera natural de resolver problemas
- Isomorfismo mundo real y modelo
- Permite manejar complejidad

POO

- Sigue siendo modular
- Altamente extensible (en buenas prácticas)

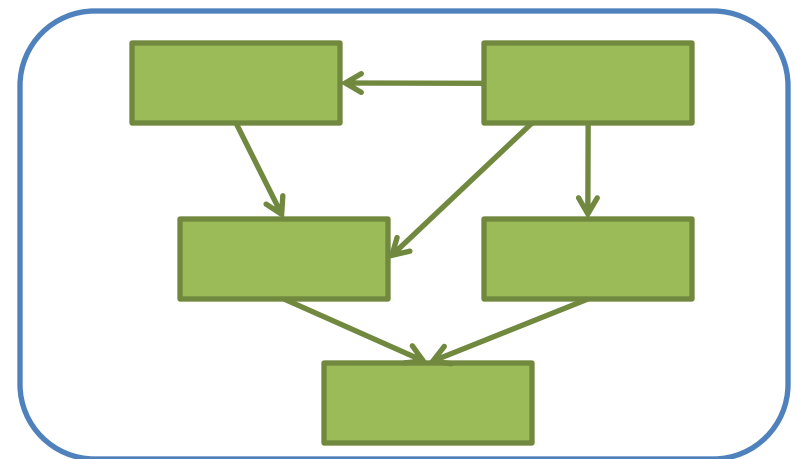
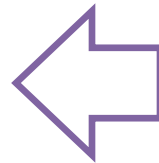
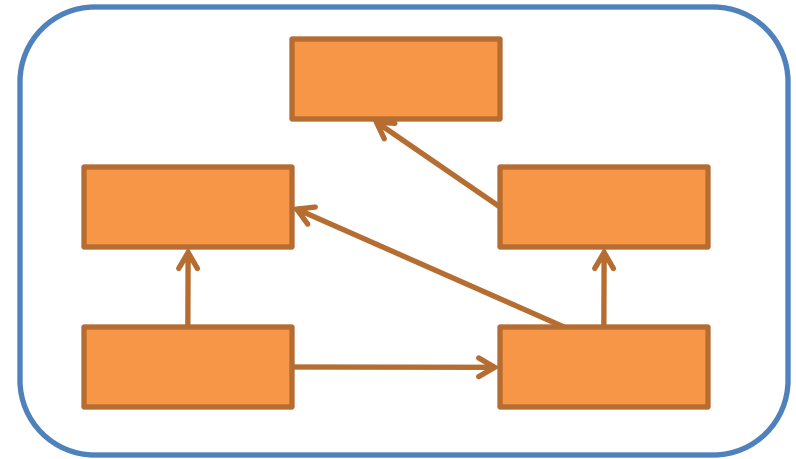
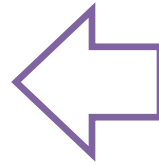
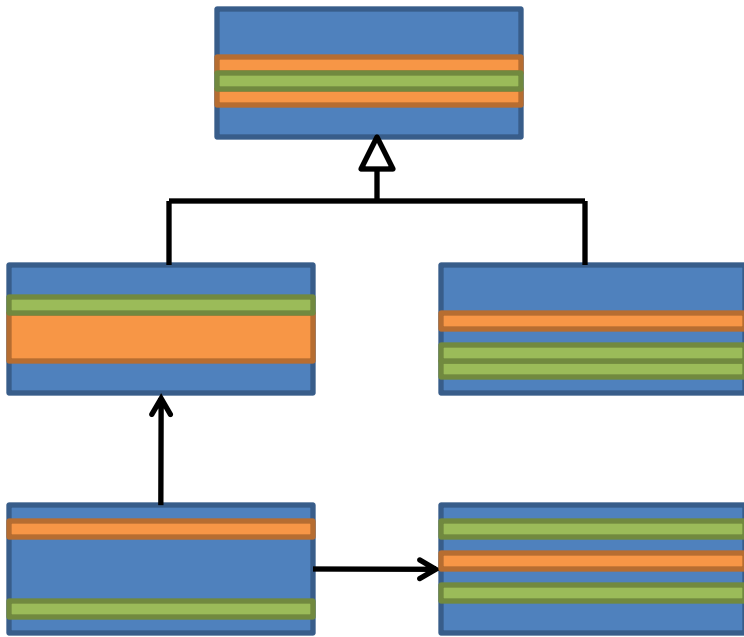
POO

Una dimensión dominante cuando diseñamos con objetos

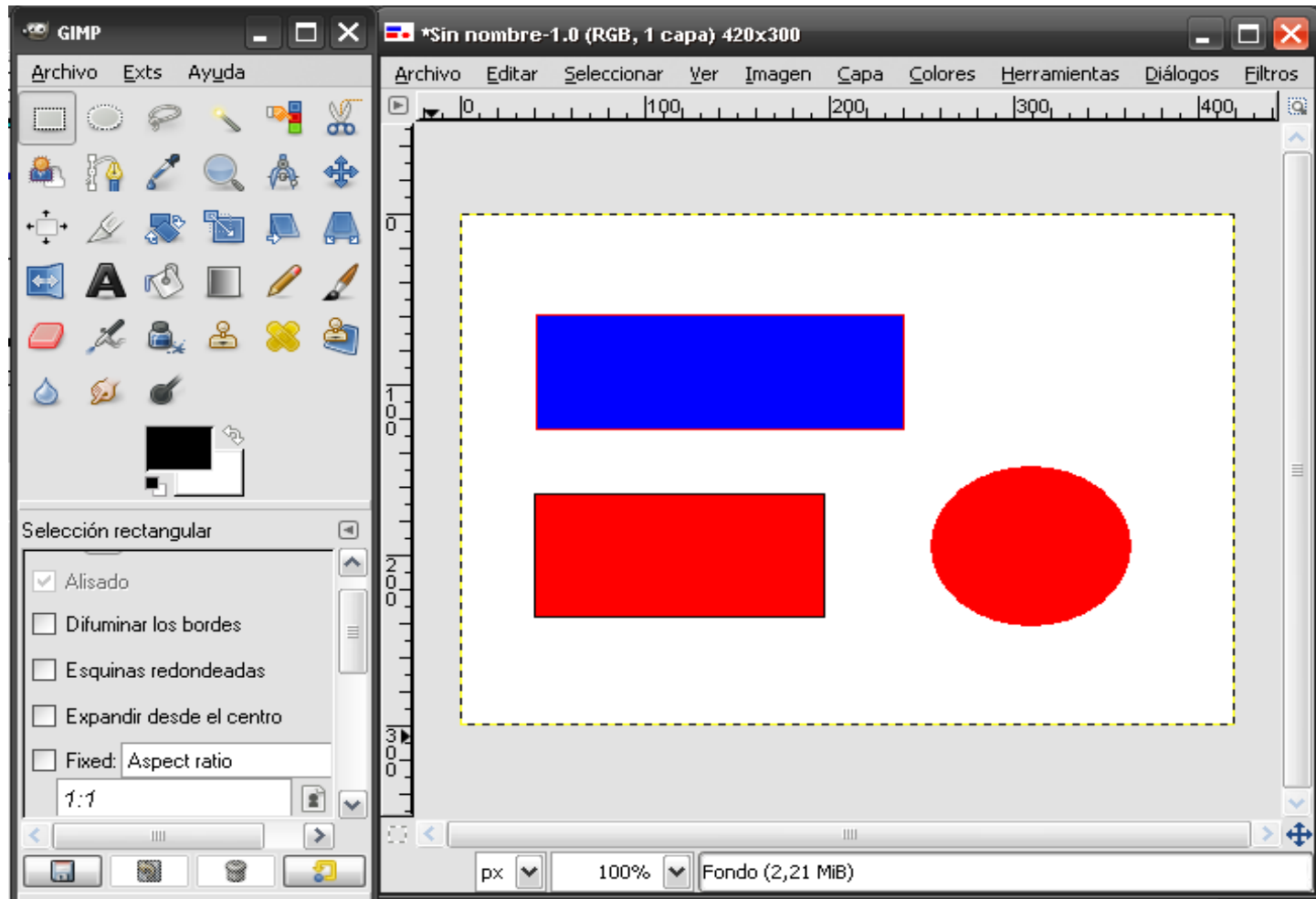


Cliente
Cuenta Corriente
Caja de ahorro
Saldo

Existen problemas que no pueden “modularizarse” correctamente con objetos?

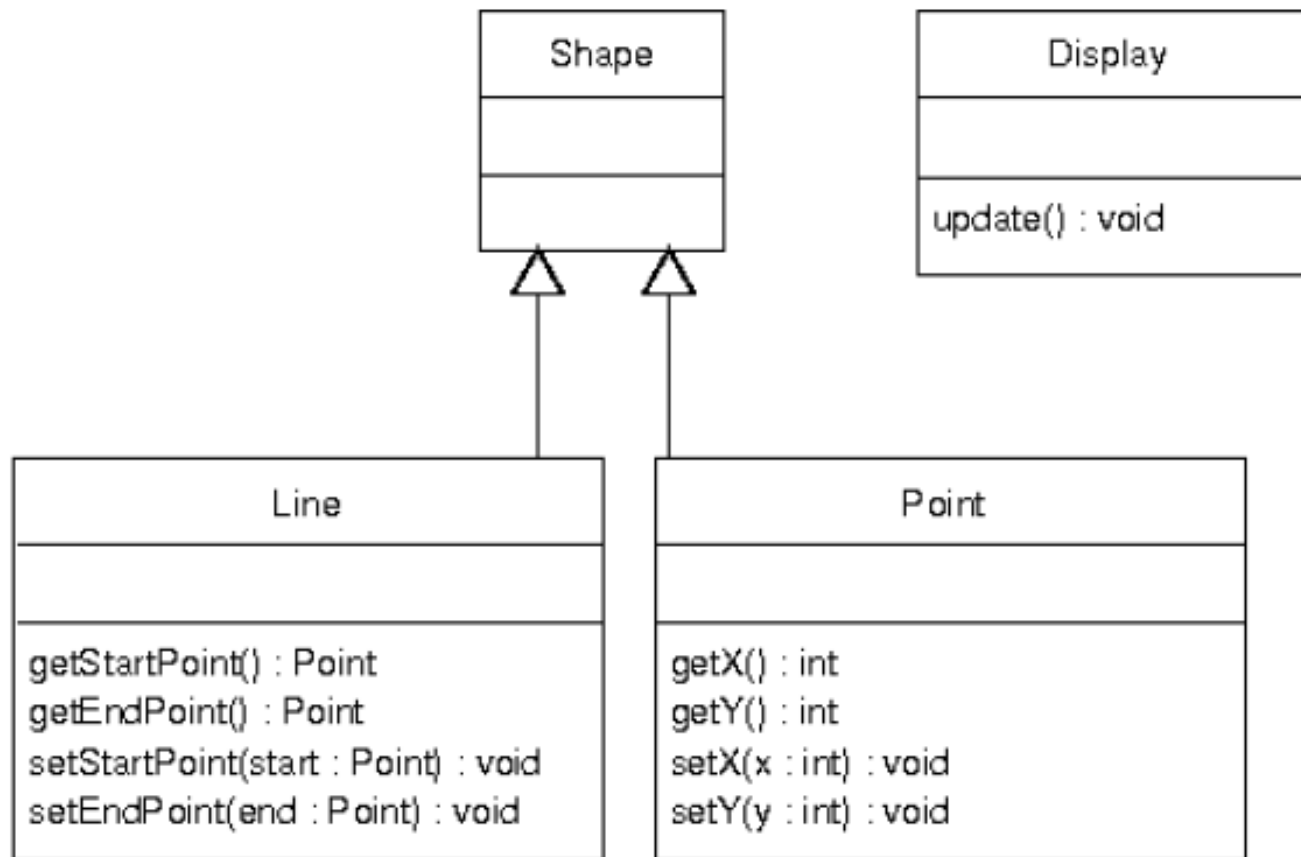


Editor Grafico, si cambia una figura de tamaño, color, posición, efectos



Se debería actualizar la pantalla

Draft 0.00001



Draft 0.00001

```
public class Point extends Shape {  
    int x=0;  
    int y=0;  
  
    public void setX(int x) {  
        this.x = x;  
        Display.update(this);  
    }  
  
    public void setY(int y) {  
        this.y = y;  
        Display.update(this);  
    }  
  
    void moveBy(int dx,int dy){  
        x = x +dx;  
        y = y +dy;  
        Display.update(this);  
    }  
  
    void toDouble(){  
        x = x * 2;  
        y = y * 2;  
        Display.update(this);  
    }  
}
```

Display.update(...)



- No es tan modular
- Intrínseco vs extrínseco

```

public class Point extends Shape {
    int x=0;
    int y=0;

    public void setX(int x) {
        this.x = x;
        Display.update(this);
        Store.save(this);
    }

    public void setY(int y) {
        this.y = y;
        Display.update(this);
        Store.save(this);
    }

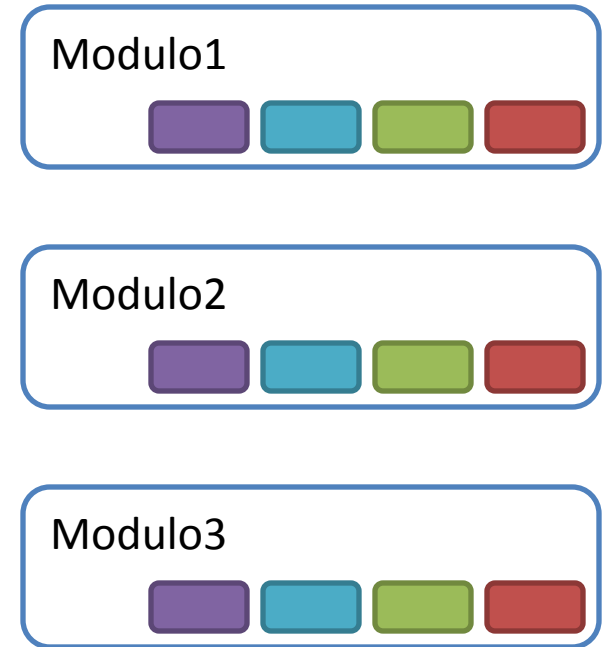
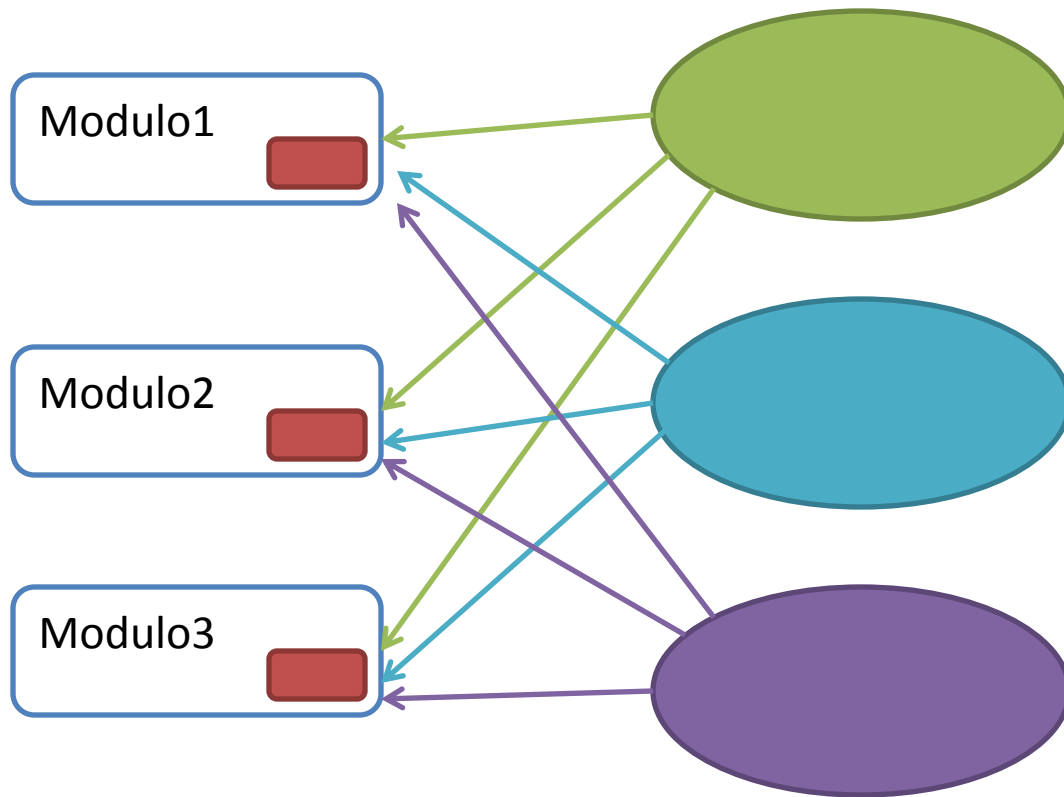
    void moveBy(int dx,int dy){
        x = x +dx;
        y = y +dy;
        Display.update(this);
        Store.save(this);
    }
}

```

Corresponde a Point
guardar y actualizar ?

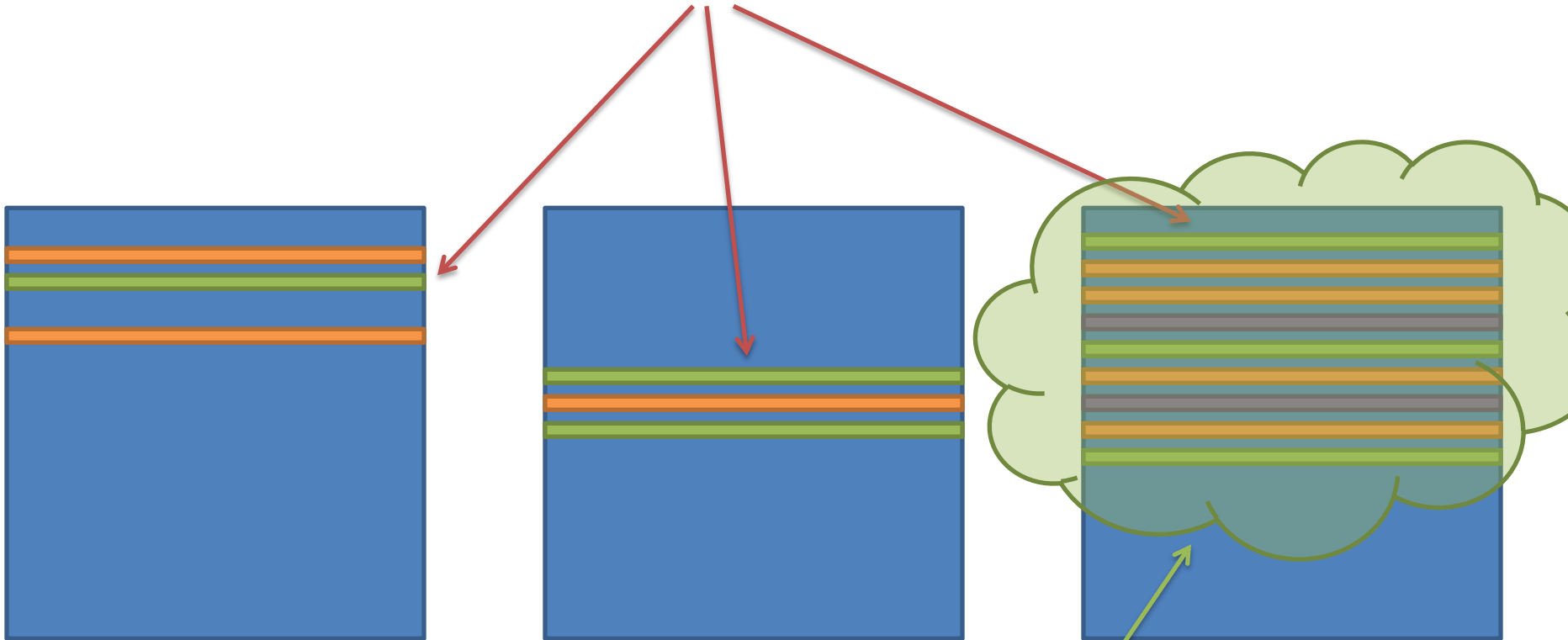
Son responsabilidades
intrínsecas ?

Diferentes concerns



Problemas de modularidad

Code Scattering



Code tangling

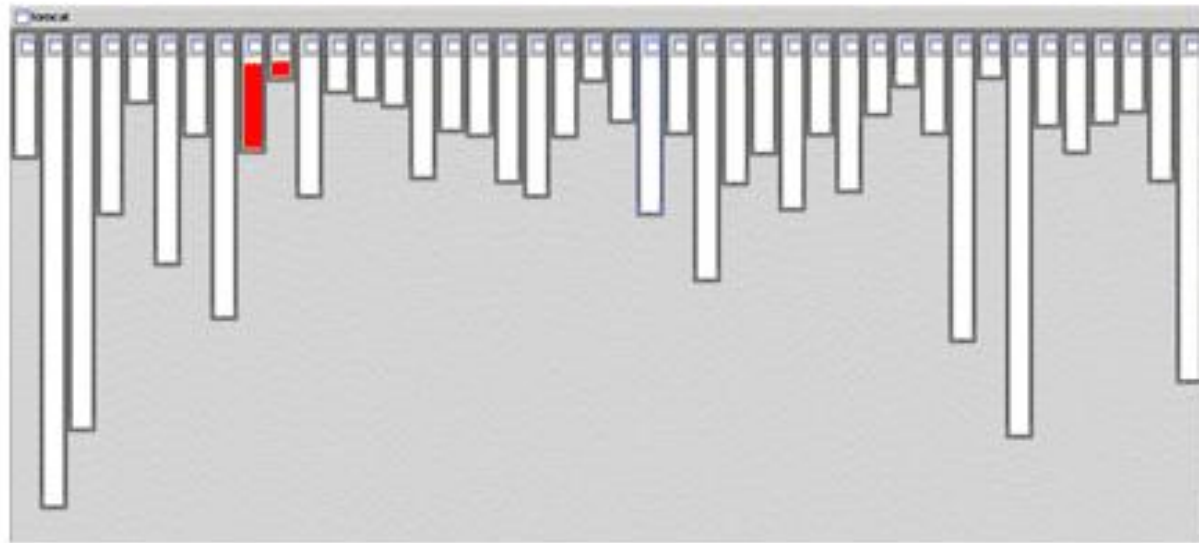
```
public class Banco {  
    // declaraciones  
  
    public double procesarDebito(long cuentaId, double monto) {  
        // apertura de transaccional  
  
        try {  
            // recupero de la cuenta  
            // validaciones de negocio  
            // logica de negocio asociada al débito  
            // persistencia del nuevo estado  
            // audito el movimiento  
            // cierre exitoso de la transacción (commit)  
            return nuevo saldo cuenta;  
        } catch (Exception e) {  
            // auditar la exception  
            // cierre anormal de la transacción (rollback)  
            // relanzamiento de la excepcion para las capas superiores  
        }  
    }  
}
```

Transaccionalidad

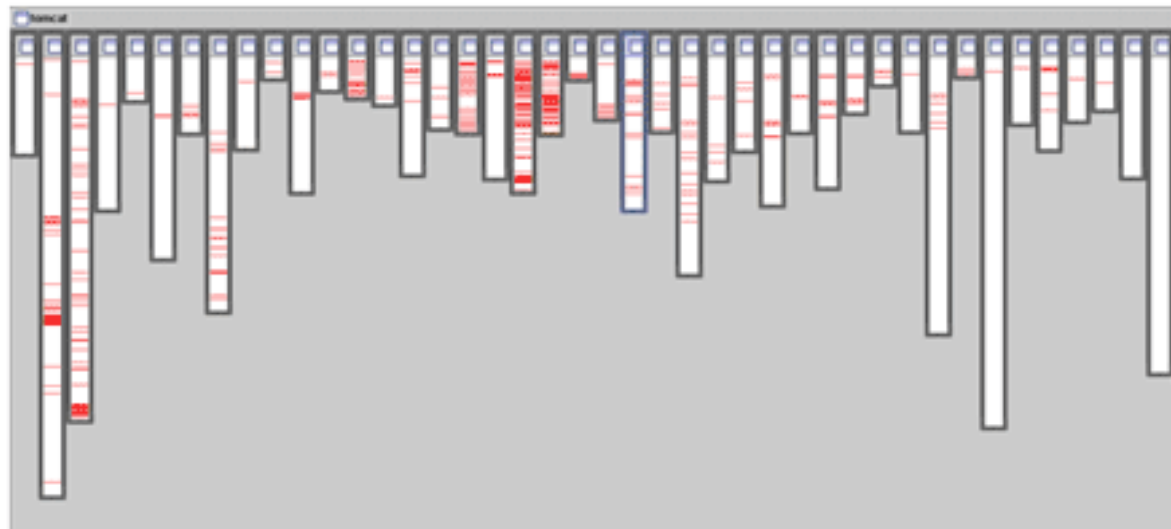
Auditoria

Persistencia

URL pattern matching
en org.apache.tomcat



Logging
en org.apache.tomcat



La falta de modularidad - consecuencias

Código disperso

Código entrelazado

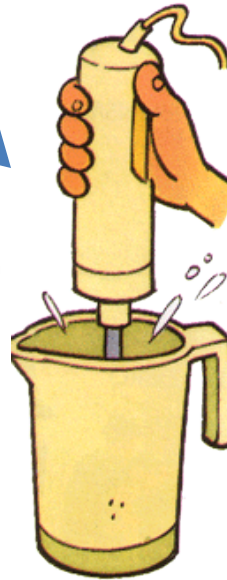
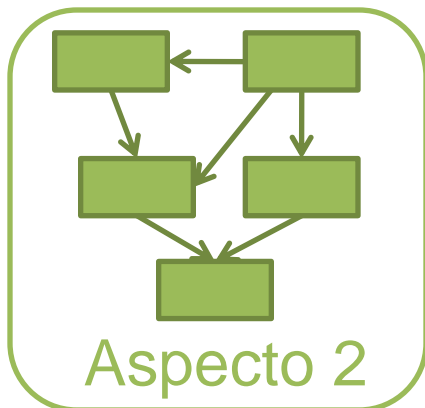
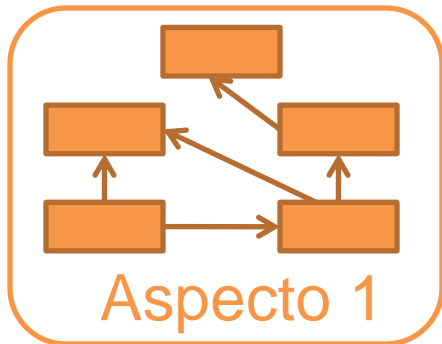
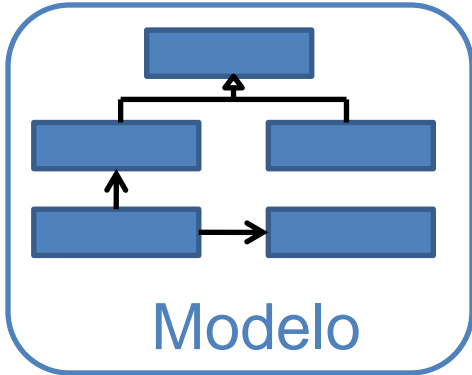
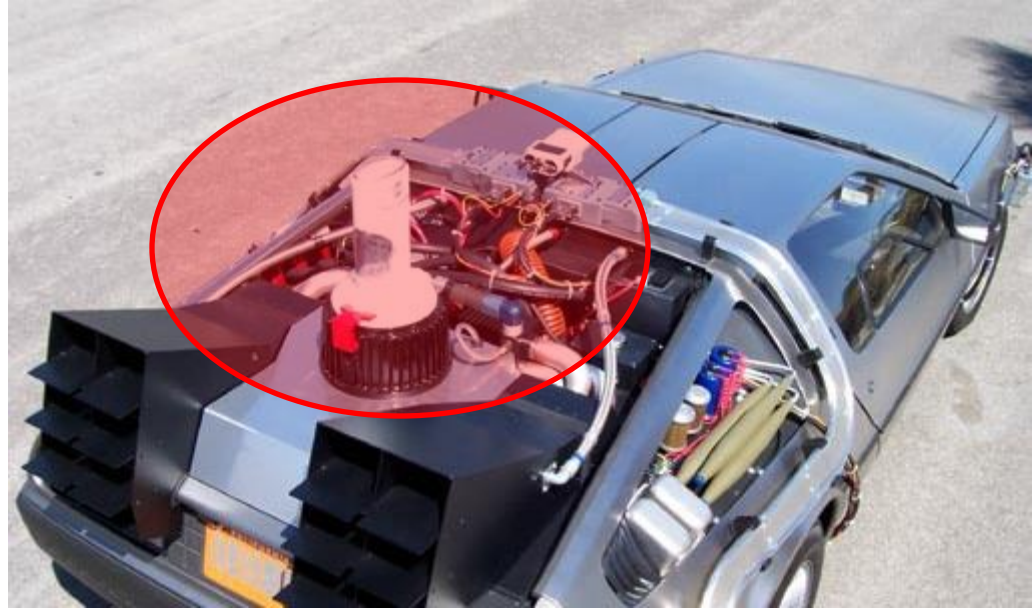
Falta de expresividad

Código independiente mezclado

Reduce la mantenibilidad (concerns)

Reduce la mantenibilidad (sistemas)

Batidora



Solución final



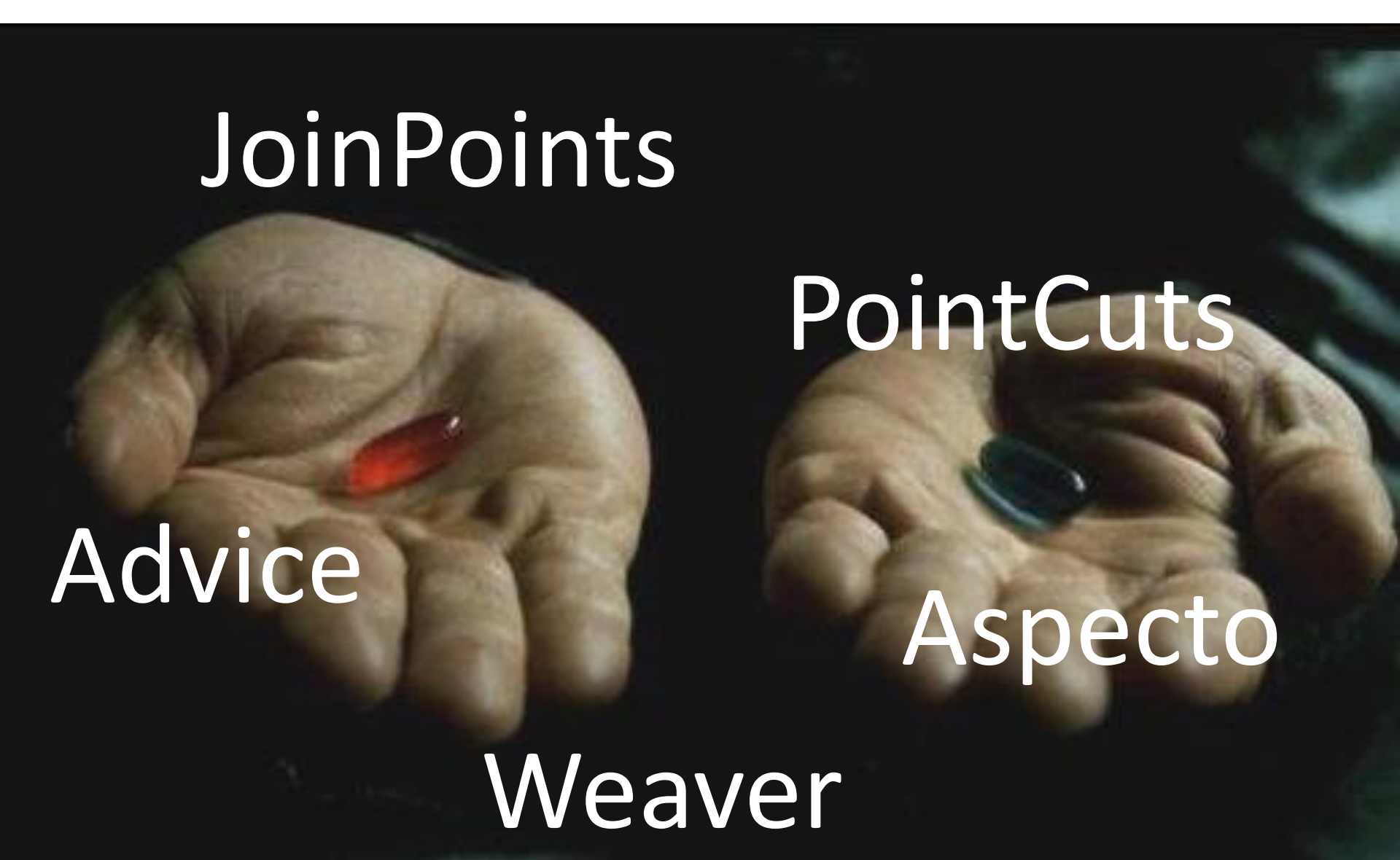
JoinPoints

PointCuts

Advice

Aspecto

Weaver



JoinPoints

Puntos del sistema donde aplico un aspecto

```
public class PersonServiceImpl implements PersonService {
```

```
    private PersonRepository personRepository;
```

```
    private BuilderFactory factory;
```

```
    private DTotoEntityMapper beanMapper;
```

```
    public void addPerson(final IPersonDTO personDTO) {
```

```
        Person person = new Person();
```

```
        // map DTO to entity
```

```
        beanMapper.map(personDTO, person);
```

```
        // save
```

```
        personRepository.save(person);
```

```
    }
```

```
    public void delete(final Serializable id) {
```

```
        Person person = personRepository.findById(id);
```

```
        personRepository.delete(person);
```

```
    }
```

```
    public void update(final IPersonDTO personDTO) {
```

```
        Person person = personRepository.findById(personDTO.getId());
```

```
        if (person == null)
```

```
            throw new IllegalAccessException();
```

```
        beanMapper.map(personDTO, person);
```

```
        personRepository.update(person);
```

Call de un método

Ejecución de un método

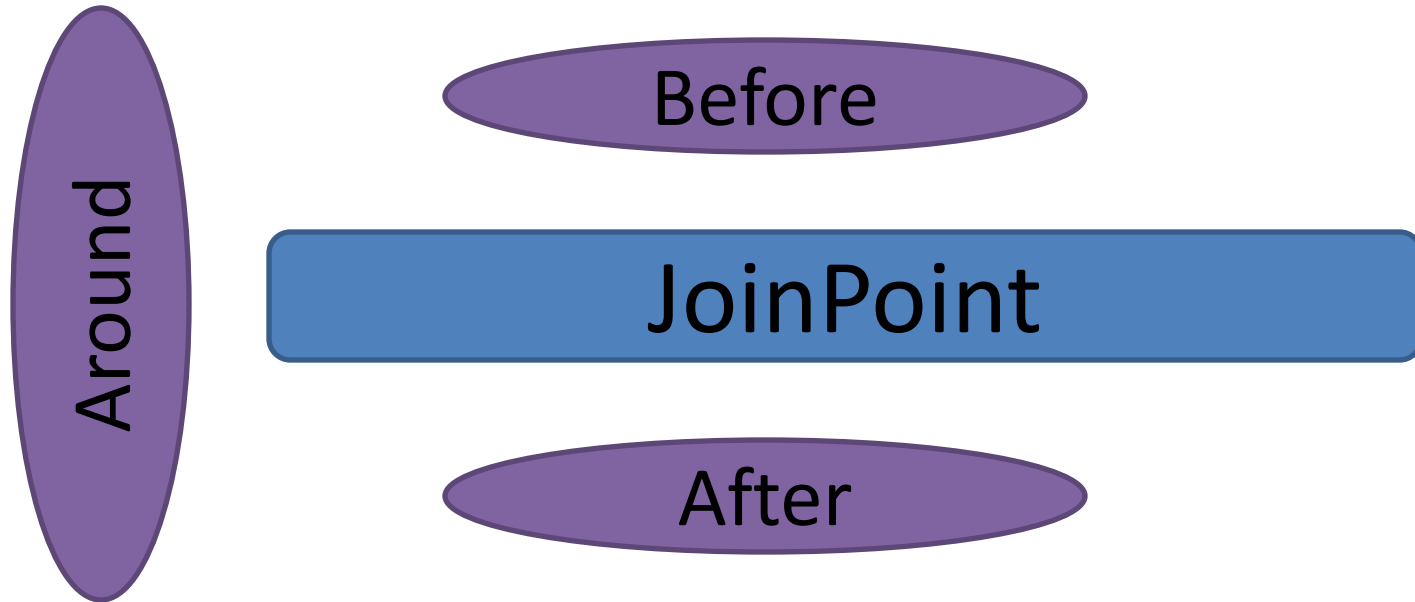
Raise Exception

Pointcuts

- Especificación de que JoinPoints aplican
- Cuantificación de condiciones
 - * Todas los raise de `IllegalAccessError`
 - * Todas los call method `*test*`
 - * Todas las implementaciones `getInstance()`*
 - * En los `@transactional`

Advice

Donde se aplica la acción en un joinpoint



Aspecto

Nuevo “modulo” formado por pointcuts mas advices.

Weaver

Mezcla aspectos con el modelo/sistema

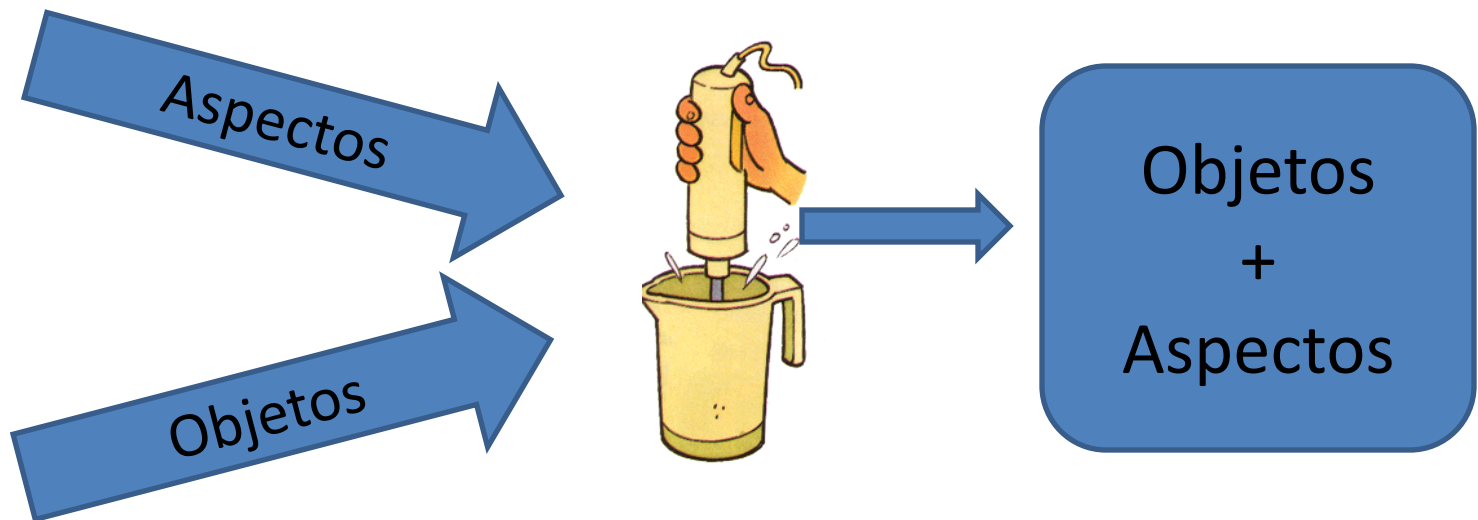
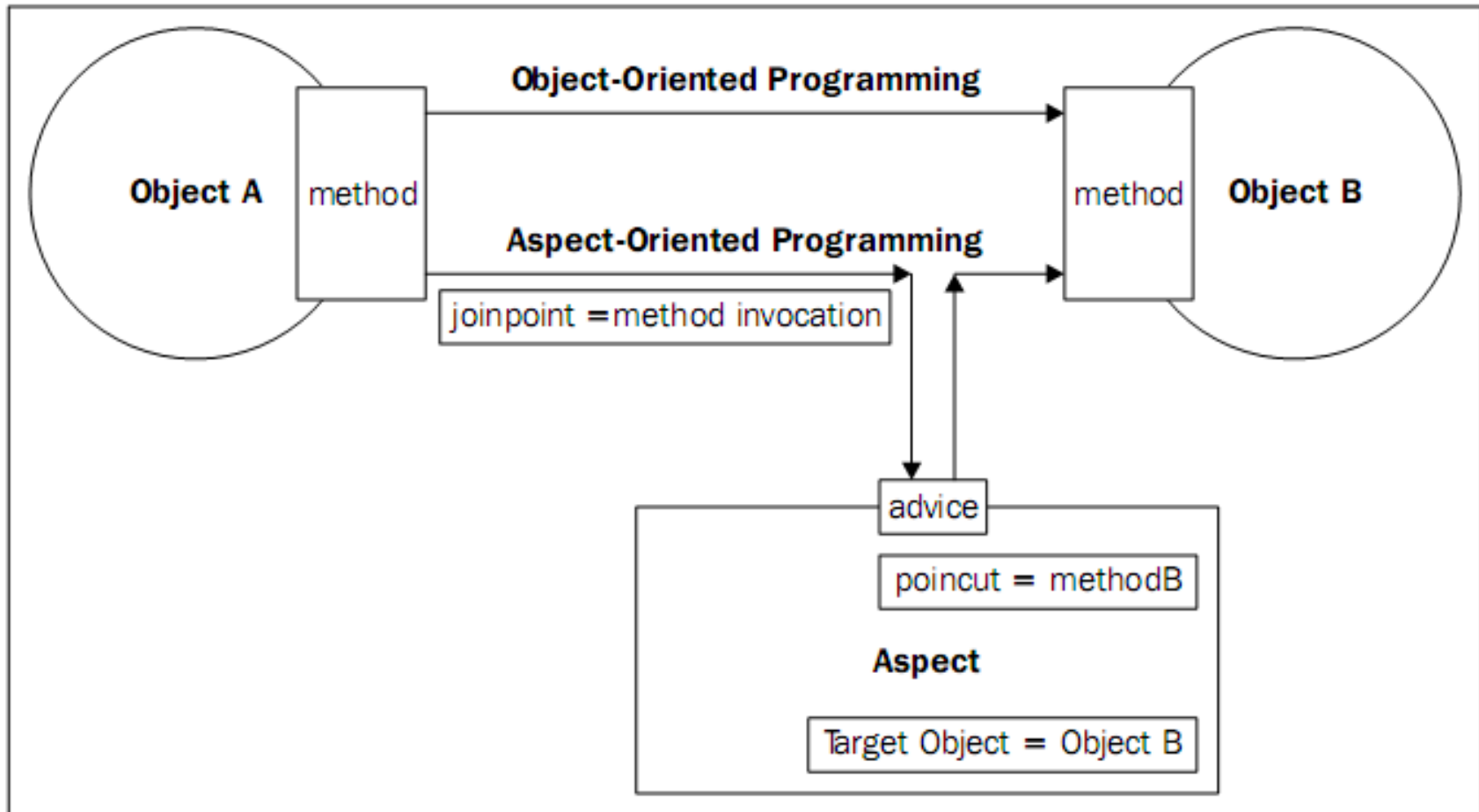
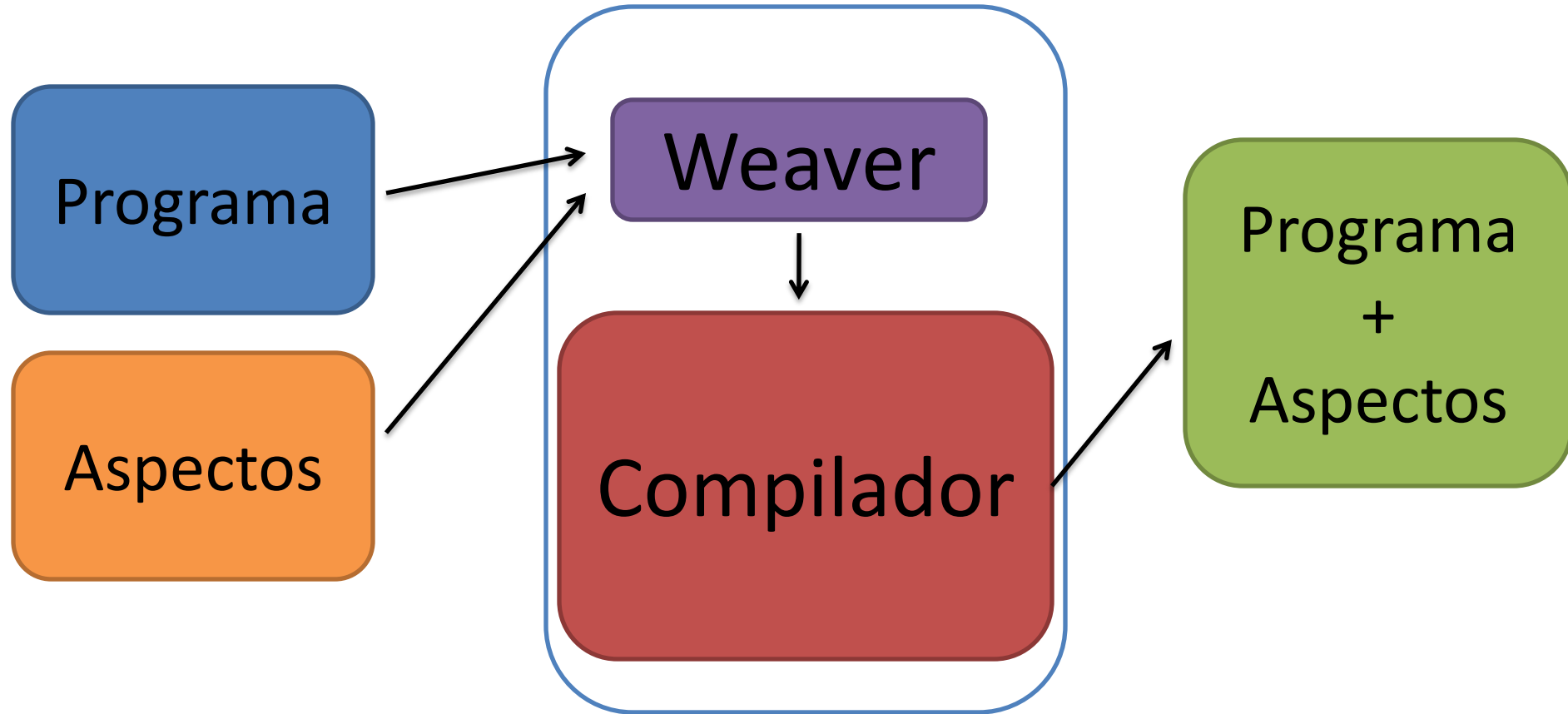


Grafico de spring con objetos

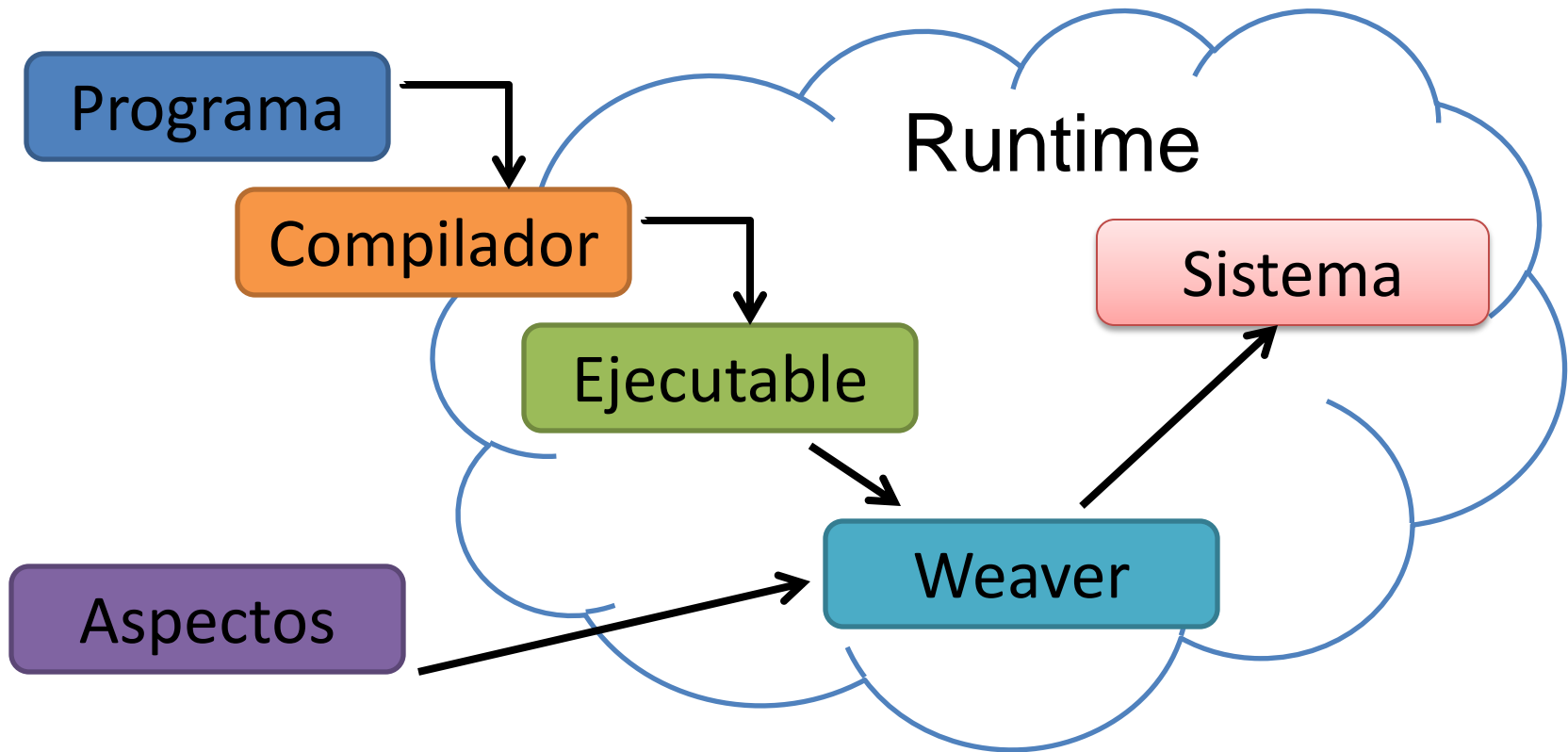


Weaver estático



- Inyección código en tiempo de compilación
AspectJ

Weaver dinámico



- Proxy
- On-The-Fly class generator

Servicios de Soporte

TX

Notificadores

Logging

Seguridad

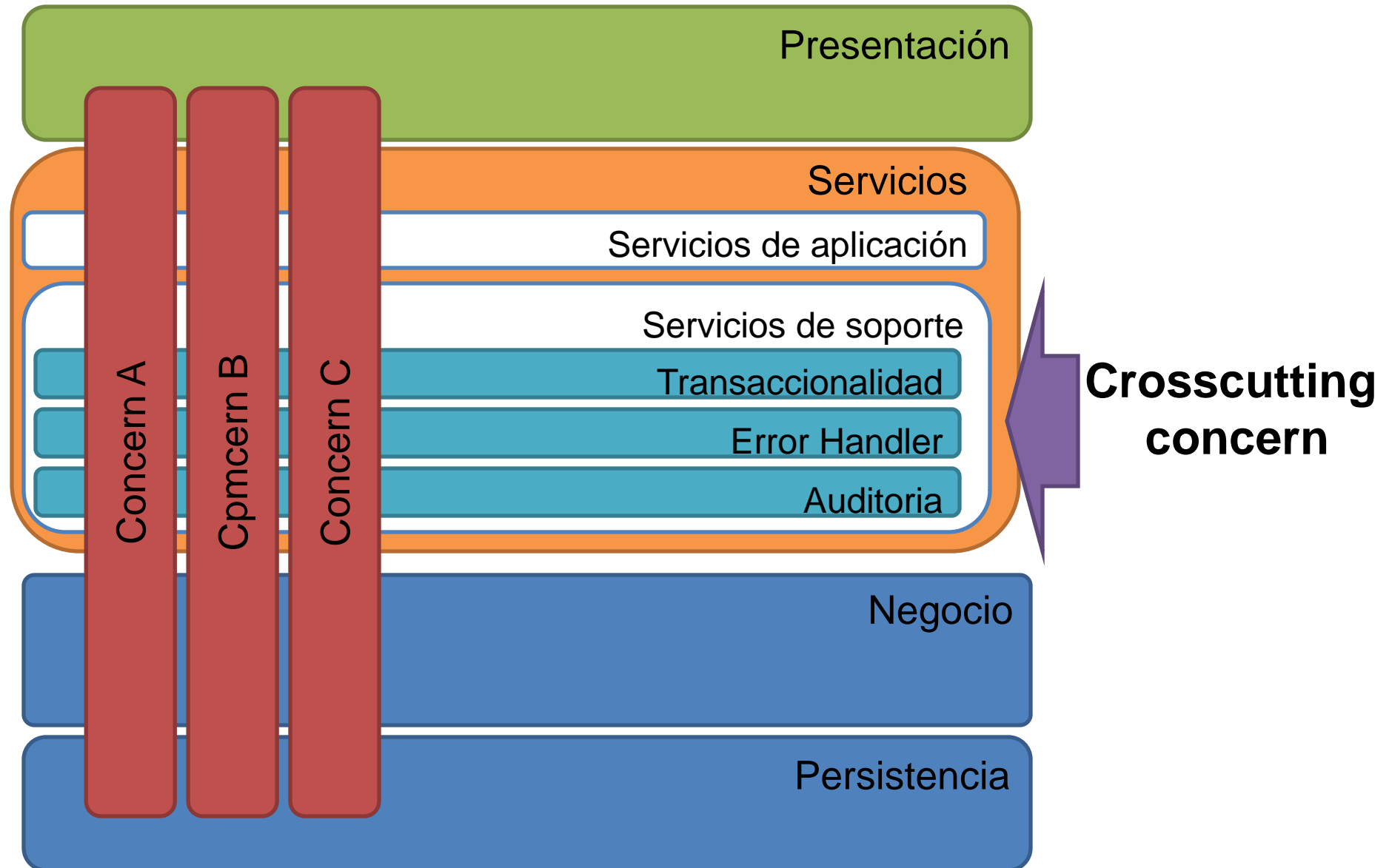
Auditoria

Cola de Mensajes

Persistencia

Manejo de Errores

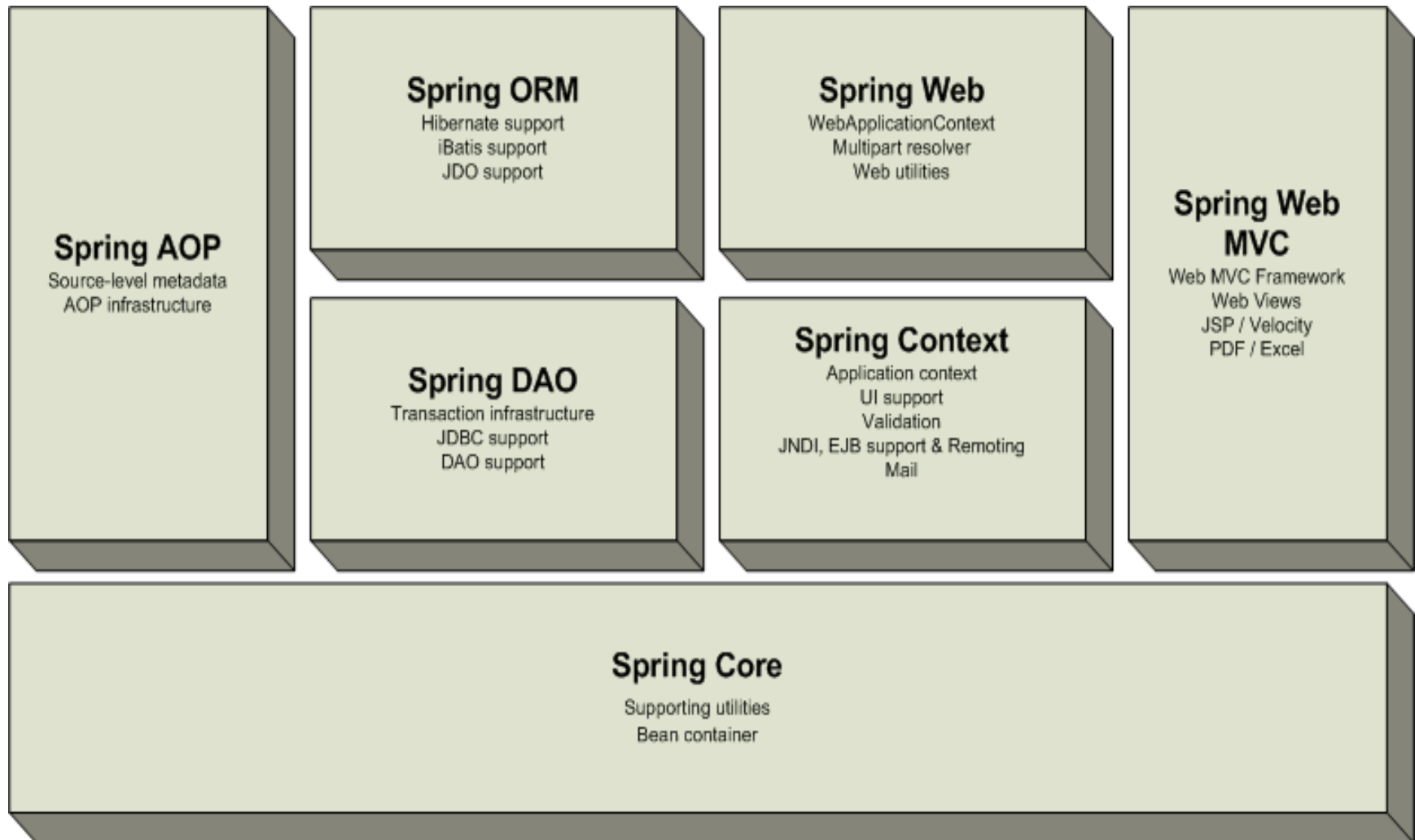
Servicios de Soporte



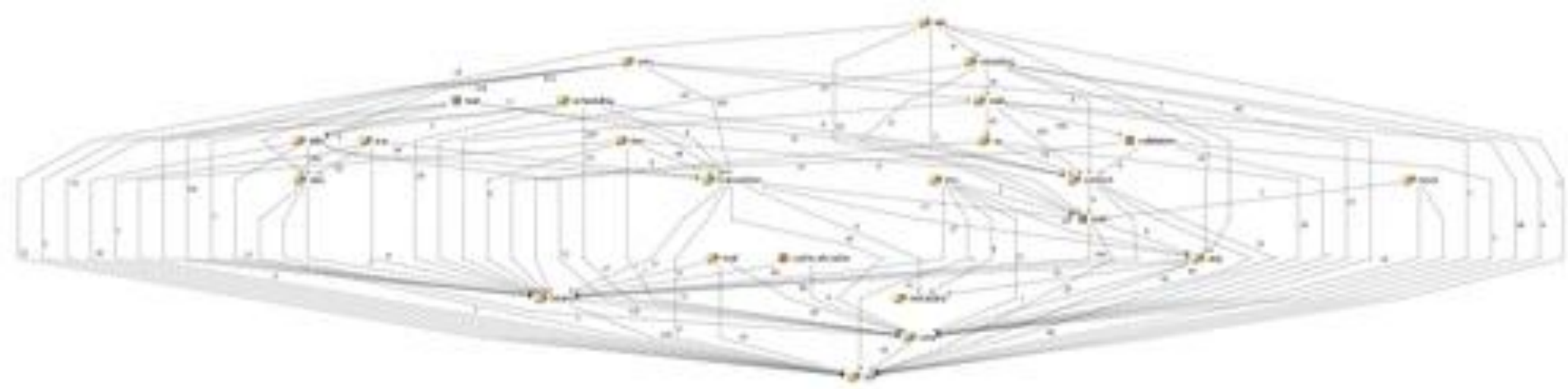


Spring

Spring



Spring



-139 packages

-Integración con tecnología del mercado

Spring

- Contenedor de Beans
- IoC
- Inyección de dependencias

IoC (construcción)

```
class B{  
    //constructor vacio  
    public B(){  
    }  
}
```

```
class C extends B{  
    //constructor vacio  
    public C(){
```

```
class A{  
    private B b;  
    //constructor vacio  
    public A(){  
        // b es un colaborador interno y se inicializa dentro del constructor  
        b = new B();  
    }  
}
```

IoC (construcción)

```
class A{  
    private B b;  
    //constructor vacio  
    public A(B otherb){  
        b = otherb;  
    }  
}
```

Seam

```
public void main(String[] args) {  
    C c = new C();  
    A a = new A(c);  
}
```

Contenedor de Beans / IoC

```
package examples;

public class Persona{
    private double fechaNacimiento;

    private String nombre;

    private Persona padre = null;

    public Persona() {}

    public Persona(double año, String nombre) {
        this.fechaNacimiento = año;
        this.nombre = nombre;
    }

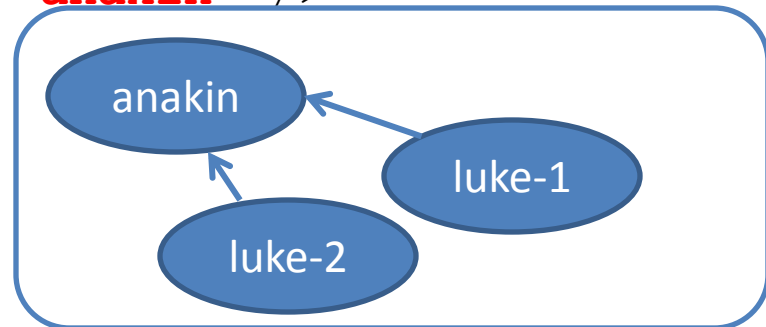
    public Persona(double año, String nombre, Persona padre) {
        this.padre= padre;
        this.fechaNacimiento = año;
        this.nombre = nombre;
    }
}
```

Contenedor de Beans / IoC

```
<bean id="anakin" class="examples.Persona">
  <constructor-arg index="0" value="958"/>
  <constructor-arg index="1" value="Anakin Skywalker"/>
</bean>

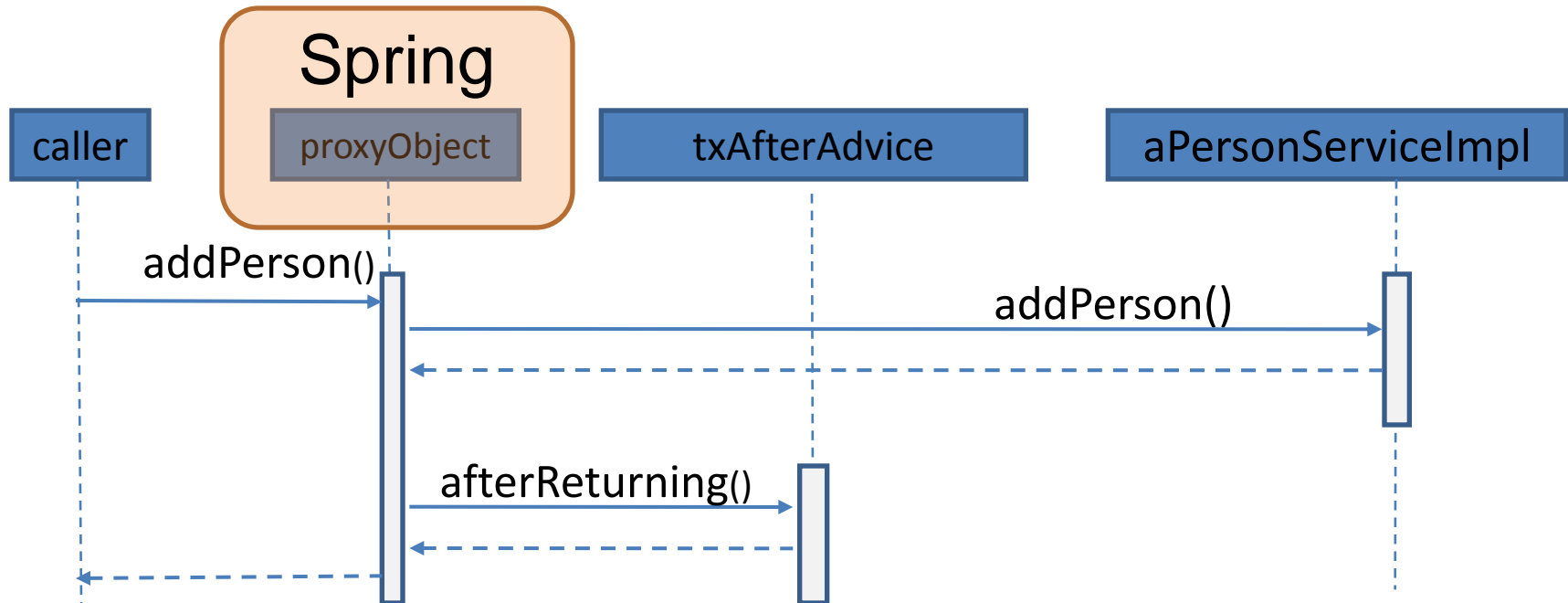
<bean id="luke-1" class="examples.Persona">
  <constructor-arg index="0" value="981"/>
  <constructor-arg index="1" value="Luke Skywalker"/>
  <constructor-arg index="1" ref="anakin"/>
</bean>

<bean id="luke-2" class="examples.Persona">
  <property name="padre" ref="anakin"/>
</bean>
```

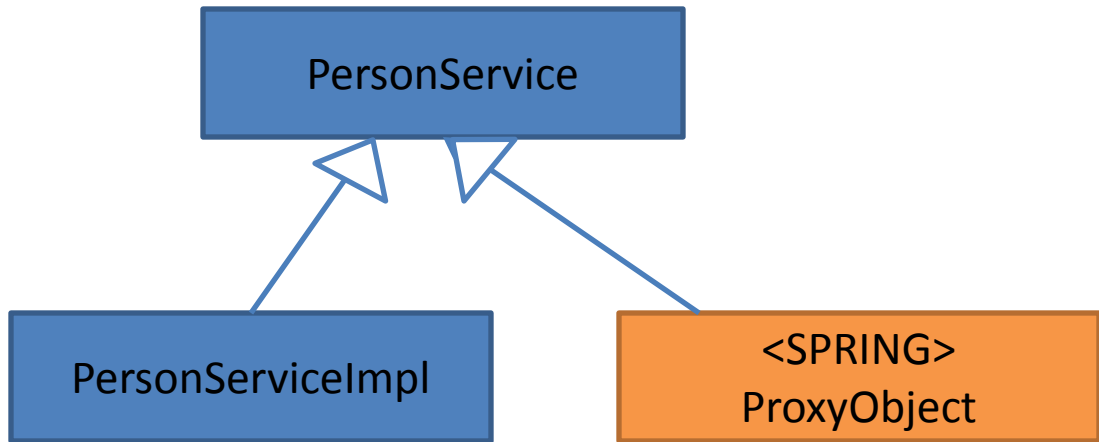


Spring AOP

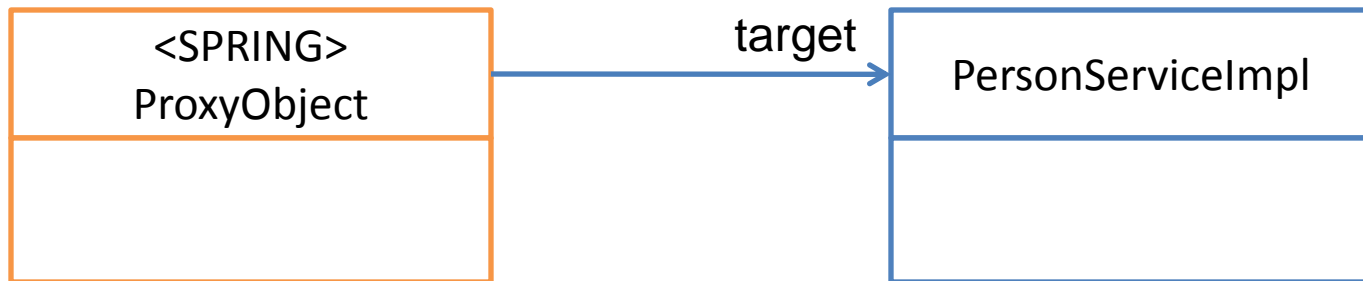
- Implementación mediante proxy



Proxy



Mismo protocolo







Este documento se distribuye
bajo la siguiente licencia
[Creative Commons Argentina](#)
[Atribución - NoComercial 2.5](#)