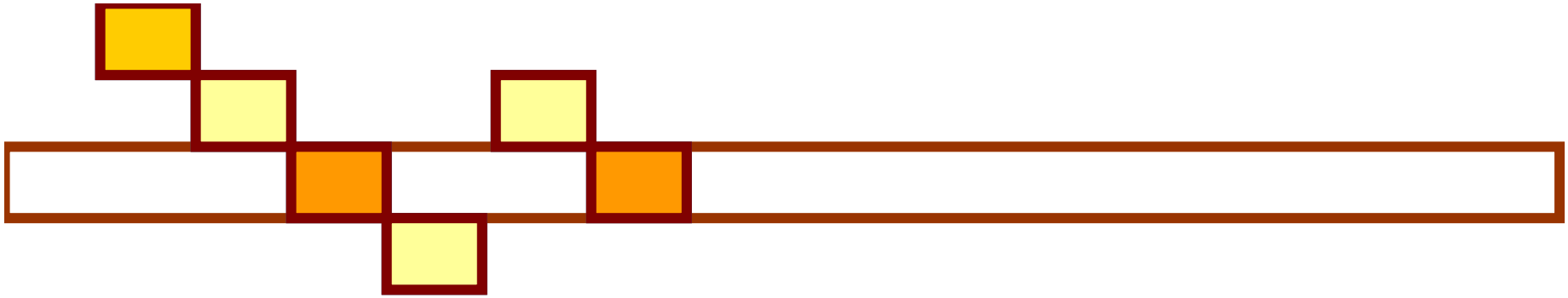


Adoptando el Paradigma de la Programación Orientada a Atributos

Raúl Marticorena, Carlos López y Carlos Pardo
Área de Lenguajes y Sistemas Informáticos
Departamento de Ingeniería Civil
UNIVERSIDAD DE BURGOS
{rmartico, clopezno, cpardo}@ubu.es

Teruel, 16 al 18 de julio

XIII Jornadas de Enseñanza Universitaria de la Informática

Tabla de Contenidos

Introducción

- Paradigma de programación orientada a objetos (POO) establecido como **contenido troncal / obligatorio:**
  - **con ventajas inherentes**
  - **nuevos problemas que necesitan nuevas soluciones**

- Problemas:
  - aspectos semánticos comunes repartidos por el código
    - Programación Orientada a Aspectos
  - semántica específica de la aplicación o dominio en ciertos elementos
    - Programación Orientada a Atributos (POA)

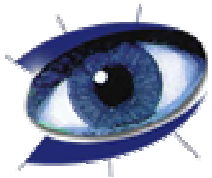
- Solución POA:
  - entrelazado de la lógica de negocio con cuestiones semánticas
  - utilizando **"marcas"** en el código
    - código declarativo embebido en código imperativo
  - ejecución de código a partir de dichas **"marcas"**

Introducción

- Idea no tan nueva...
  - Ej: preprocesadores en C
  - Ej: JML en Java para implementar Diseño por Contrato
  - Ej: XDoclet para componentes distribuidos EJB
  - Ej: estereotipos y valores etiquetados en UML

- Adoptado en la resolución de nuevos problemas
  - Ej: pruebas, servicios web, componentes distribuidos, etc.

- ***Cuestiones***
  - ¿Necesaria su inclusión?
  - ¿Cómo realizar dicha inclusión?
  - ¿Necesario un nuevo esfuerzo?



Problemas

- Concepto **aparentemente trivial**
- Pero sus usos y utilidades **no son tan fáciles de entender**

Contexto Inicial

- Lenguajes de programación
  - Imperativos en programación de nivel I
    - Primer curso: C
  - Orientados a Objetos en programación de nivel II y III
    - Segundo y tercer curso: Java
  - C#, VB.NET y Java en trabajos final de carrera en tercer curso
  - C++, VB .NET y Java en segundo ciclo
- Lenguajes de modelado
  - UML desde el segundo curso hasta quinto

- Aparece el concepto de atributo en plataformas de amplio uso:
  - en Java 1.5 denominado **anotación**
  - en .NET denominado **atributo**

- **Utilizado en nuevas herramientas, entornos, etc.**

Primera Experiencia: Uso de Atributos (I)

- Inclusión de pruebas
  - Asignatura de programación de nivel II
  - Usando JUnit (versión 3.x)
    - Diseño basado en patrones
    - Reflexión e introspección (***convención de nombres***)
- Ej:

```
public void setUp(){ // Inicializa el estado antes de cada test.
        // ...
}
public void testComprobar(){
        // ...
}
public void testExcepcion(){
        try{
                // código que genera excepción
                fail(); // provocar fallo
        }
        catch(ExcepcionEsperada es){
                // ok
        }
}
```

Primera Experiencia: Uso de Atributos (II)

- Uso de anotaciones
  - JUnit (versión 4.x)
    - Uso de anotaciones con sintaxis `@Anotación`
    - Sin referencia a los patrones de diseño
    - Semántica de ejecución ligada a la anotación
- Ej:

```
@Before
public void inicializar(){
}
@Test
public void comprobarPrecio(){
}
@Test(expected=ExcepcionEsperada.class)
public void exception(){
        // código que genera excepción
}
```

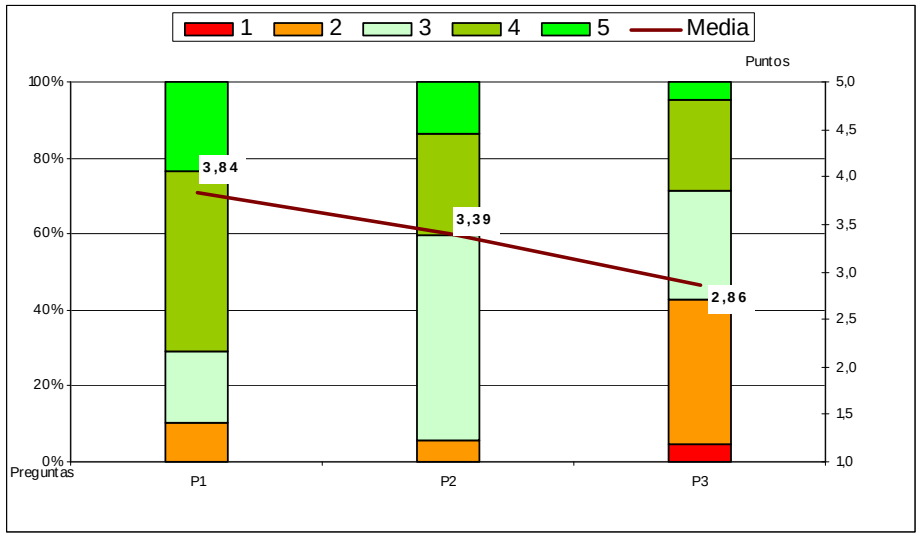
Primera Experiencia: Uso de Atributos (III)

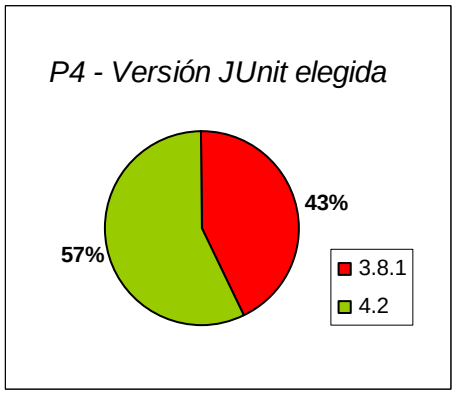
- ¿Cómo se ha abordado su introducción?
  1. Presentación breve del concepto de anotación
  2. Enumeración de anotaciones en JUnit y semántica asociada
  3. Se muestra algún ejemplo simple
  4. Se aporta un código de ejemplo "con huecos" (plantilla)
  - Se omite:
    - procesado y ejecución de las anotaciones

- Práctica obligatoria: implementación de pruebas unitarias utilizando anotaciones

Primera Experiencia: Uso de Atributos (IV)

- Encuesta:
  - P1- ¿Aumentar tiempo de explicación del concepto?
  - P2 -¿Es conveniente aprender el concepto?
  - Opinión de alumnos que han usado ambas versiones:
    - P3- Tendencia a considerar más fácil o más difícil el uso de anotaciones
    - P4- Preferencia clara sobre una u otra versión





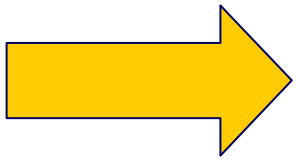
Segunda Experiencia: Definición de Atributos (I)

- Asignatura: de diseño y mantenimiento del software, 5º curso
  - métricas, patrones de diseño (PD), pruebas y refactorizaciones

- Doble actividad:
  - definición de atributos asociados a patrones (PD)
  - traducción a código de conceptos de UML: valor etiquetado y estereotipos

Segunda Experiencia: Definición de Atributos (II)

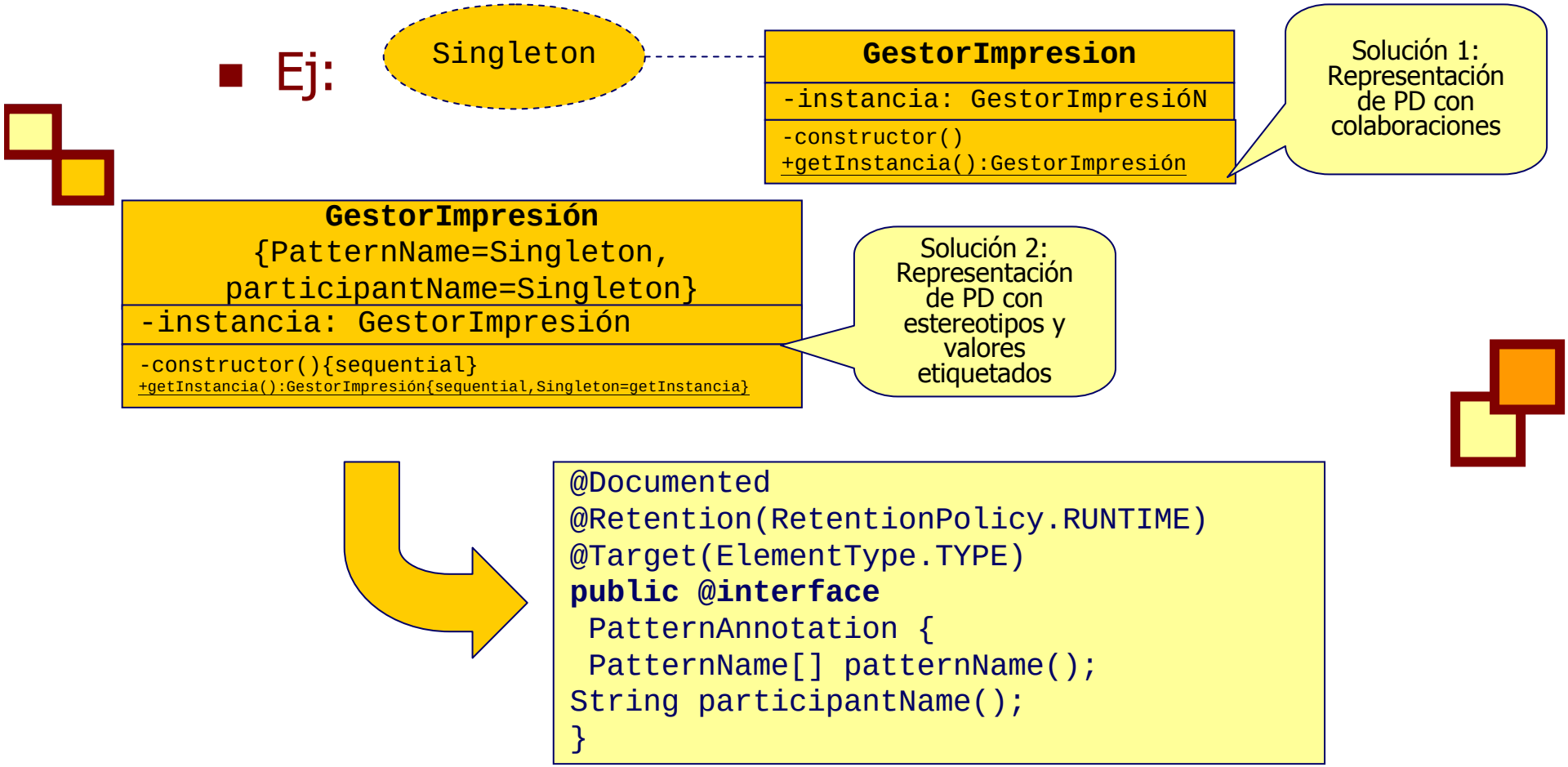
- Definición de Atributos
  - Sesiones prácticas guiadas:
    - extracción del diseño de tres aplicaciones a partir del código fuente
    - uso de herramienta WOP de identificación de patrones
      - precisión baja de la herramienta



  - Solución: incluir la semántica con atributos en el propio código
    - Definidas e implementadas por los alumnos
    - Requiere conocimientos avanzados sobre anotaciones

Segunda Experiencia: Definición de Atributos (III)

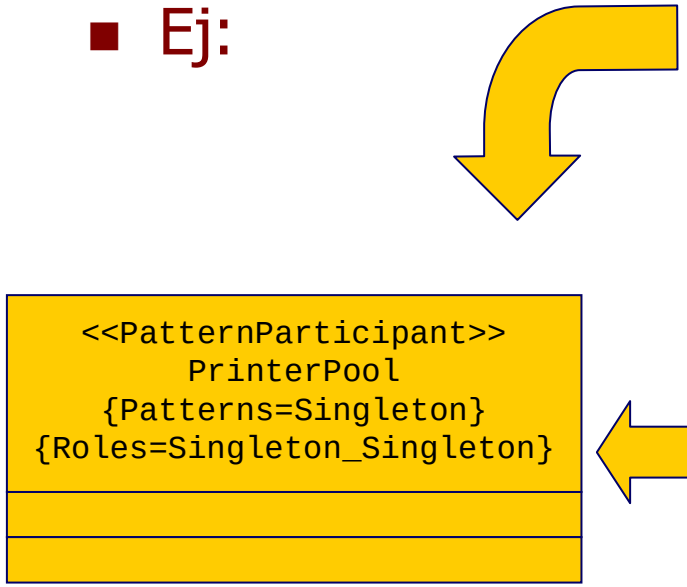
- Ej:




```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface
 PatternAnnotation {
 PatternName[] patternName();
String participantName();
}
```

Segunda Experiencia: Definición de Atributos (IV)

- Traducción UML a Java
  - Añadir funcionalidad a una herramienta CASE UML
  - Traducir los estereotipos y valores etiquetados a código
- Ej:

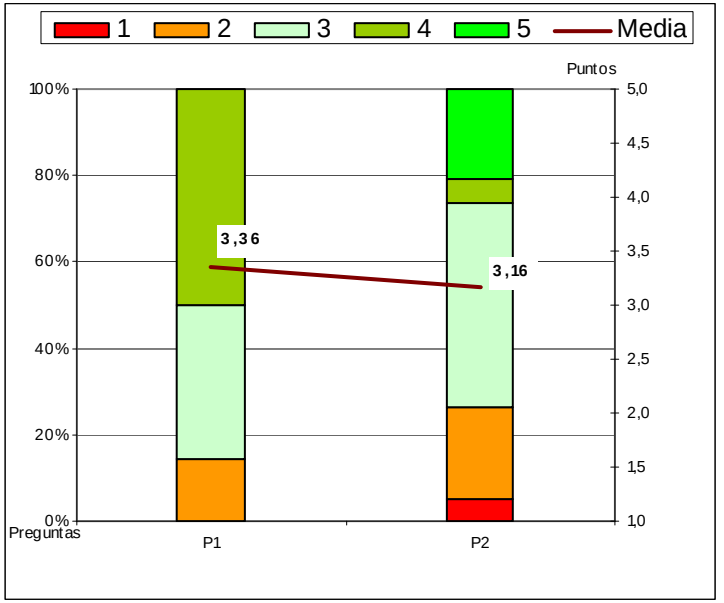



```
import java.lang.annotation.*;
@Documented
@Retention(RetentionPolicy.RUNTIME
)
@Target(ElementType.TYPE)
public @interface
 PatternParticipant {
 String[] Patterns();
 String[] Roles();
}
```

```
@PatternParticipant (
 Patterns={"Singleton"},
 Roles={"Singleton_Singleton"})
public class PrinterPool{
}
```

Segunda Experiencia: Definición de Atributos (V)

- Evaluación:
  - P1- ¿Clarifican las anotaciones el diseño?
  - P2- ¿Identificación de PD mejora la comprensión?



- Comentarios:
  - Interés elevado en añadir contenido semántico
    - Ven adecuado el uso de atributos para este fin
  - No tan clara la utilidad de la semántica en la identificación de patrones

Tercera Experiencia: Uso Avanzado y Evaluación (I)

- Asignatura: de sistemas distribuidos, 5º curso
- Prácticas con componentes Entreprise JavaBeans
  - En curso previos con la especificación 2.1
    - Interfaces, descriptor de despliegue XML
  - Versión actual 3.0 → Uso intensivo de atributos
- Ej: Message Driven Bean

EJB 2.1

```
public class MdbBean
        implements
        MessageDrivenBean,
        MessageListener {
        // lógica de negocio de acceso
        // a la cola de mensajes
}
```

Descriptor XML de despliegue

Descriptor XML de despliegue propio de la plataforma J2EE

EJB 3.0

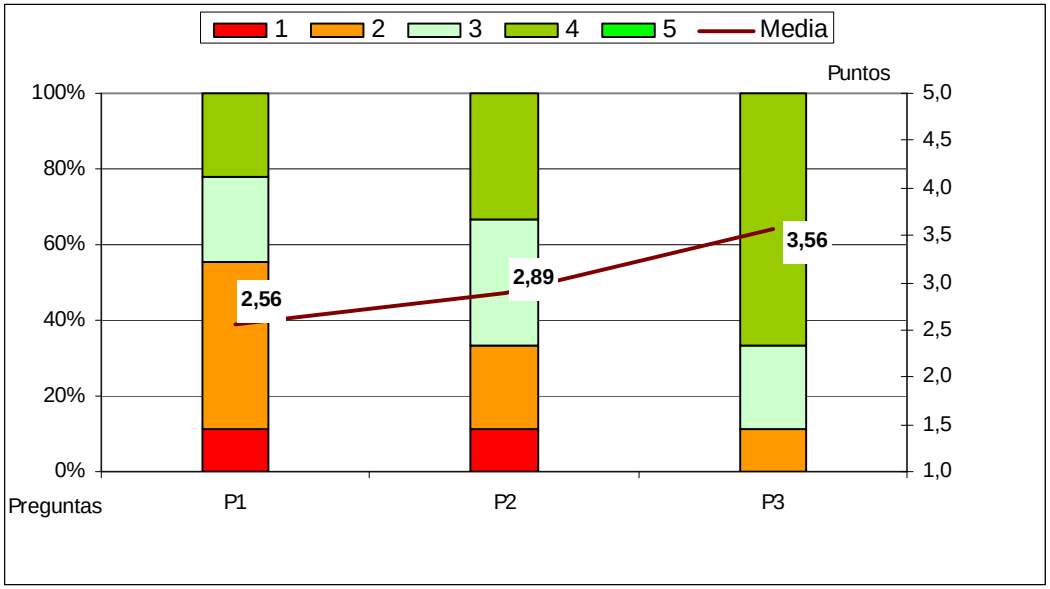
```
@MessageDriven(activationConfig = {
@ActivationConfigProperty(propertyName =
"destination", propertyValue =
"ColaEjemplo"),

@ActivationConfigProperty(propertyName =
"destinationType", propertyValue =
"javax.jms.Queue")
    }
)
public class MessageDrivenBean
implements MessageListener {
          // cuerpo de la clase
}
```

Tercera Experiencia: Uso Avanzado y Evaluación (II)

- Práctica:
  - 1. Implementación en las dos especificaciones
    - plataforma JOnAS (EJB 2.1) y EasyBeans (EJB 3.0)
  - 2. Evaluación comparativa desde el punto de vista del *"responsable de la empresa de I+D+i"*
    - Toma de decisión de futuro

- Evaluación
  - P1- ¿Aumentar tiempo de explicación del concepto?
  - P2- ¿Es conveniente aprender el concepto?
  - P3- ¿Más fácil el desarrollo con atributos?



Conclusiones

- Necesaria una introducción teórica y práctica en asignaturas de POO de cursos iniciales
  - El simple uso sin explicaciones adicionales puede llevar a cierta desorientación
  - Adecuado coordinar su inclusión con asignaturas de ingeniería del software
- El concepto se asimila mejor en segundos ciclos o últimos cursos de ciclo superior
- Realizar prácticas comparativas entre ambas soluciones clarifica el concepto
- Resultados extrapolables a paradigmas cercanos a la POO
  - Programación Orientada a Aspectos

Líneas de Trabajo Futuro

- Observar el efecto de su introducción en primeros cursos
  - Suposición: agilizar el desarrollo usando atributos y la resolución de problemas más complejos en cursos superiores
- Empezar a plantear la introducción del concepto de aspecto:
  - Siguiendo un proceso similar
  - Aplicando los resultados obtenidos con sus pros y contras
- Aplicado a otras plataformas como .NET