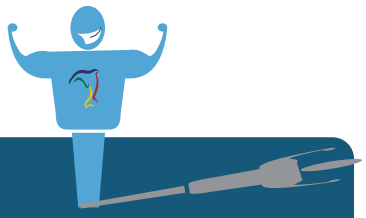
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

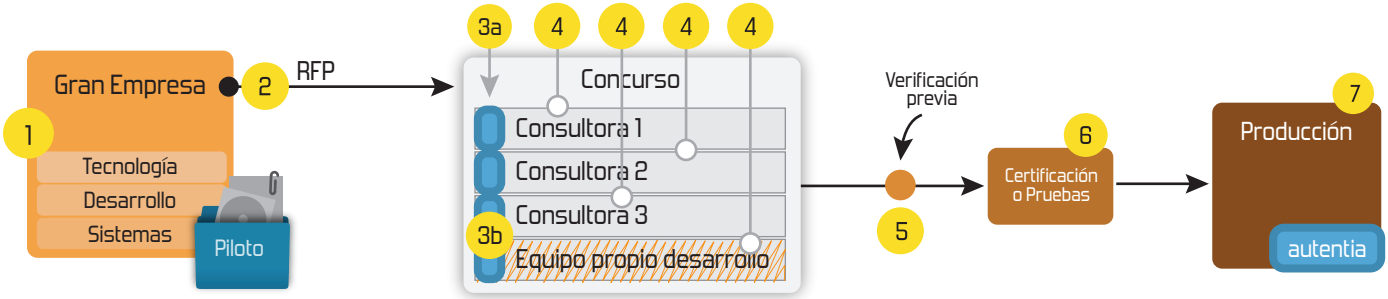
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Como nota, las cuatro anotaciones pertenecen al paquete **javax.persistence**.

- **@Embeddable** : Con esta anotación, indicamos que la clase puede ser "integrable" dentro de una entidad.
- **@Embedded** : Con esta anotación, indicamos que el campo o la propiedad de una entidad es una instancia de una clase que puede ser integrable. Es decir, para que funcione, el campo que hayamos anotado como @Embedded, debe corresponderse con una clase que tenga la anotación @Embeddable.
- **@AttributeOverrides** : Con esta anotación, podemos sobreescribir o redefinir el mapeo de campos o propiedades de tipo básico. Esta anotación recibe un array de @AttributeOverride
    - **@AttributeOverride**: Con esta anotación, podemos redefinir el mapeo de tipos básicos. Su uso es @AttributeOverride(name="atributoDeLaClaseEmbebida", column = @Column(name="nombreColumnaEnLaTabla")),
- **@AssociationOverrides** : Con esta anotación, podemos sobreescribir o redefinir el mapeo de campos o propiedades complejos. Esta anotación recibe un array de @AssociationOverride
    - **@AssociationOverride**: Con esta anotación, podemos redefinir el mapeo de tipos básicos. Su uso es @AssociationOverride(name="atributoDeLaClaseEmbebida",joinColumns=@JoinColumn(name="columna_id")), donde el primer parámetro es el nombre del atributo de la clase embebida que estamos redefiniendo, y el seguno parámetro es un **@JoinColumn**, en donde podemos especificar por que campo hacemos el join.

4. Ejemplos

Ahora que ya sabemos para que se usa cada anotación, vamos a ver unos ejemplos de como podemos usarlas. Los ejemplos buscan más una manera de mostrar el uso de las anotaciones de una forma práctica que mostrar un escenario completamente real.

Tenemos una clase Coche.java en donde vamos a incrustar dos atributos de la misma clase Asiento.java. La definición principal de las dos clases es:

view plain print ?

```
01. package com.adictosaltrabajo.tutoriales;
02.
03. import javax.persistence.Entity;
04. import javax.persistence.GeneratedValue;
05. import javax.persistence.Id;
06.
07. @Entity
08. public class Coche {
09.
10.     @Id
11.     @GeneratedValue
12.     private Integer id;
13.
14.     Motor motor;
15.
16.     public Coche() {
17.         // TODO Auto-generated constructor stub
18.     }
19.
20.     public Integer getId() {
21.         return id;
22.     }
23.
24.     public void setId(Integer id) {
25.         this.id = id;
26.     }
27.
28.     public Motor getMotor() {
29.         return motor;
30.     }
31.
32.     public void setMotor(Motor motor) {
33.         this.motor = motor;
34.     }
35. }
```

```
view plain  print  ?
01. package com.adictosaltrabajo.tutoriales;
02.
03.
04. public class Asiento {
05.
06.     int pesoMaximoEnKg;
07.
08.     //Clase compleja
09.     Cinturon cinturon;
10.
11.     public Asiento() {
12.         // TODO Auto-generated constructor stub
13.     }
14.
15.     public int getPesoMaximoEnKg() {
16.         return pesoMaximoEnKg;
17.     }
18.
19.     public void setPesoMaximoEnKg(int pesoMaximoEnKg) {
20.         this.pesoMaximoEnKg = pesoMaximoEnKg;
21.     }
22.
23.     public Cinturon getCinturon() {
24.         return cinturon;
25.     }
26.
27.     public void setCinturon(Cinturon cinturon) {
28.         this.cinturon = cinturon;
29.     }
30. }
```

Si ahora queremos que las dos clases sean una única entidad, utilizamos las anotaciones que hemos comentado anteriormente: @Embeddable y @Embedded. NOTA: solo se muestra el código referente a los cambios realizados.

```
view plain  print  ?
01. @Entity
02. public class Coche {
03.
04.     @Id
05.     @GeneratedValue
06.     private Integer id;
07.
08.     Motor motor;
09.
10.     @Embedded //Anotamos para indicar que es una propiedade embebid
11.     Asiento asientoConductor;
12.
13.     public Coche() {
14.         // TODO Auto-generated constructor stub
15.     }
16.     ......................
17. </pre>
```

```
view plain  print  ?
01. import javax.persistence.Embeddable;
02.
03. @Embeddable  //Anotamos para indicar que es una clase que se puede
04. public class Asiento {
05.
06.     int pesoMaximoEnKg;
07.
08.     //Clase compleja
09.     Cinturon cinturon;
10.
11.     public Asiento() {
12.     ......................
```

¿Que ocurre si ahora queremos meter otro asiento?

```
view plain  print  ?
01.  @Embedded
02.  Asiento asientoConductor;
03.
04.  @Embedded
05.  Asiento asientoCopiloto;
06.
07.  public Coche() {
08.  ......................
```

pues hibernate nos lanzará una excepción (org.hibernate.MappingException) indicando que hay columnas repetidas

Para solucionarlo, lo que tenemos que hacer es indicar a Hibernate como queremos realizar el mapeo de la clase usando las dos anotaciones que nos faltan por usar del tutorial ;), @AttributeOverrides y @AssociationOverrides

La clase Asiento.java, tiene dos atributos, "pesoMaximoEnKg" que es de tipo básico (int) y "cinturon" que pertenece a un tipo complejo Cinturon. Entonces hemos de indicar como mapear los dos atributos de forma que pertenezcan a columnas diferentes en la tabla de la baes de datos.

```
view plain  print  ?
01.  @Embedded
02.  @AttributeOverrides(  {
03.      @AttributeOverride(name="pesoMaximoEnKg", column = @Column(name="pesoMa
04.  }
05.  @AssociationOverrides({
06.      @AssociationOverride(name="cinturon",joinColumns=@JoinColumn(name="cint
07.  })
08.  Asiento asientoConductor;
09.
10.  @Embedded
11.  @AttributeOverrides(  {
12.      @AttributeOverride(name="pesoMaximoEnKg", column = @Column(name="pesoMa
13.  }
14.  @AssociationOverrides({
15.      @AssociationOverride(name="cinturon",joinColumns=@JoinColumn(name="cint
16.  })
17.  Asiento asientoCopiloto;
18.
19.  public Coche() {
20.      // TODO Auto-generated constructor stub
21.  }
22.  ......................
```

5. Conclusiones

Hemos visto en el tutorial que sólo con el uso de anotaciones, somos capaces de incrustar clases dentro de entidades para que se comporten con una única identidad. Esto nos puede servir para agrupar varias clases con un contexto similar, para hacer más legible el código y para muchas cosas más.

Para cualquier comentario, duda o sugerencia, tenéis el formulario que aparece a continuación.

Un saludo.

COMENTARIOS