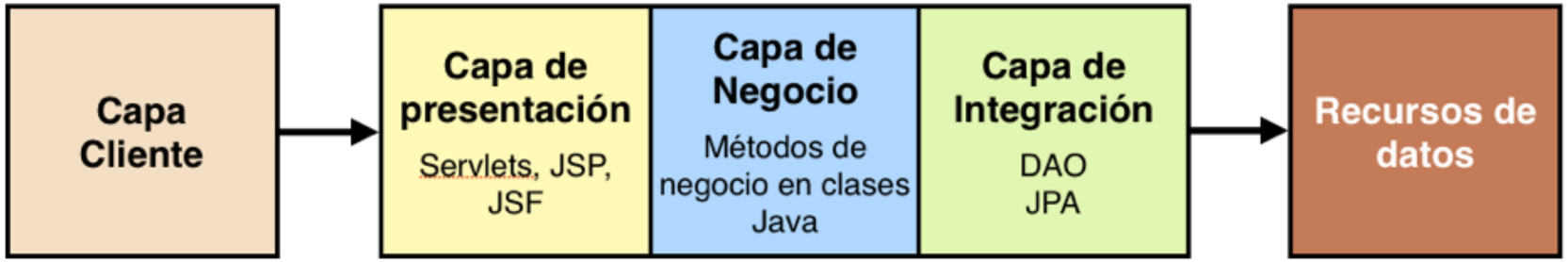
Componentes Enterprise JavaBeans

Sesión 1 - Introducción a los componentes EJB

Índice

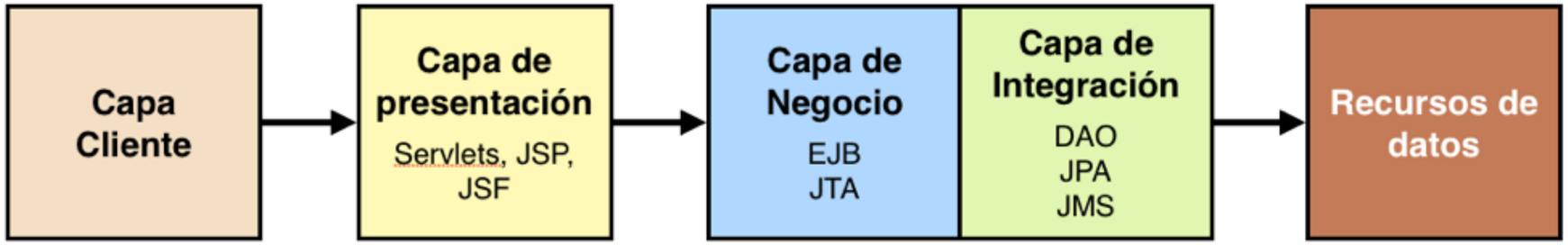
Arquitectura por capas

- Separa la lógica de la presentación
- Permite pruebas modulares
- Permite extensión incremental de la aplicación



Arquitectura componentes Enterprise JavaBeans

- Separación *física* de la capa de negocio y de la capa de presentación
- Transformación de los objetos de negocio en componentes gestionados por un contenedor EJB
- Utilización de JTA para gestionar transacciones distribuidas



Arquitectura EJB

- La arquitectura EJB se propone como un marco de computación basada en componentes distribuidos
- Historia:
  - EJB 1.0 (Abril 1998)
  - EJB 1.1 (Diciembre 1999) - Descriptores XML
  - EJB 2.0 (Agosto 2001) - Primer estándar (JSR 19)
  - EJB 2.1 (Noviembre 2003) - Interfaces locales y servicios web
  - EJB 3.0 (Mayo 2006) - POJOs y anotaciones
  - EJB 3.1 (Diciembre 2009) - EJB Lite, Singletons, invocación asíncrona

Componentes enterprise beans

- Los enterprise beans son clases Java gestionadas por el servidor de aplicaciones (contenedor EJB)
- Similares a los **beans CDI,** no los creamos llamando a `new()` sino que el contenedor los inyecta en variables anotadas
- La diferencia con CDI es que en los enterprise beans **todas las llamadas a sus métodos pasan a través del contenedor**, que proporciona un conjunto de servicios adicionales que afectan a esa invocación del cliente
- Desde el punto de vista de un cliente, un enterprise bean:
  - Proporciona acceso a un conjunto de servicios definidos por su interfaz de negocio.
  - El contenedor recubre la interfaz de negocio con un conjunto de servicios añadidos (seguridad, transaccionalidad, concurrencia, escalabilidad) que se implementan de forma transparente al desarrollador del bean

Servicios proporcionados por el contenedor EJB

- Acceso remoto de múltiples clientes (no lo vamos a ver en profundidad en el curso, porque vamos a usar servicios web JAX-RS)
- Seguridad
- Transaccionalidad distribuida con JTA
- Acceso a recursos
- Concurrencia
- Threading
- Gestión y pooling de recursos
- Persistencia
- Gestión de mensajes
- Escalabilidad

Tipos de enterprise beans

- Beans de sesión
  - Encapsula un conjunto de métodos o acciones de negocio que **son invocados de forma síncrona**
  - Ejemplos: métodos de peticiones de reserva de libros de una biblioteca o un carrito de la compra
  - En general, cualquier componente que ofrezca un conjunto de servicios (métodos) a los que se necesite acceder de forma distribuida, segura y transaccional
  - Vamos a centrarnos en este tipo de beans
- Beans dirigidos por mensajes (MDBs)
  - La comunicación con ellos no es por medio de invocaciones, sino por el envío de mensajes
  - Se encolan y se procesan de forma asíncrona
  - Ejemplo: Un MDB `ListenerNuevoCliente` que se activara cada vez que se envía un mensaje comunicando que se ha dado de alta a un nuevo cliente
  - Muy usados en arquitecturas distribuidas empresariales

Primer ejemplo: definición de un bean de sesión sin estado

Anotación @Stateless

Método de negocio

```
import javax.ejb.Stateless;

@Stateless
public class SaludoServicio {

    private String[] misSaludos = {"Hola, que tal?",
            "Cuanto tiempo sin verte", "Que te cuentas?",
            "Me alegro de volver a verte"};

    public String saludo(String nombre) {
        int random = (int) (Math.random() * misSaludos.length);
        String saludo = nombre + ", " + misSaludos[random];
        return saludo;
    }
}
```

Primer ejemplo: inyección e invocación desde un servlet

Inyección

```java
@WebServlet(name="holamundo", urlPatterns="/holamundo")
public class HolaMundoServlet extends HttpServlet {

    @EJB
    SaludoServicio saludoServicio;

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException,
                                                              IOException {
        ...
        response.setContentType("text/html");
        out.println("<HTML>");
        out.println("<BODY>");
        out.println("<h1>Stateless</h1>");
        out.println("<ul>");
        out.println("<li>" + saludoServicio.saludo(nombre) + "</li>");
        out.println("</ul>");
        out.println("</BODY>");
        out.println("</HTML");
    }
}
```

Invocación

La llamada es interceptada por el contenedor

- Con sólo anotar el método de negocio con una restricción de seguridad, la invocación deja de funcionar

```
@RolesAllowed({"usuario-saludo"})
   public String saludo(String nombre) {
        int random = (int) (Math.random() * misSaludos.length);
        String saludo = misSaludos[random];
        return nombre + ", " + saludo;
   }
```

- La llamada `saludoServicio.saludo(nombre)` no invoca el método directamente, sino a través del servidor de aplicaciones

Tipos de beans de sesión

- Beans sin estado
- Beans con estado
- Beans singleton

Beans de sesión sin estado

- Los métodos de negocio reciben datos y devuelven resultados sin modificar el estado del bean. El contenedor crea una reserva (pool) de instancias, todas ellas del mismo bean y asigna cualquier instancia a cualquier llamada de un cliente
- Si hacemos distintas llamadas desde un cliente cada vez puede ser que nos responda una instancia distinta
- Tan pronto como la invocación al método termina su ejecución, la instancia del bean está disponible para otros clientes
- Son muy escalables, el contenedor EJB no tiene que mover sesiones de la memoria a un almacenamiento secundario para liberar recursos, simplemente puede obtener recursos y memoria destruyendo las instancias
- Se usan en general para encapsular procesos de negocio. Proporcionan un conjunto de procesos relacionados con un dominio específico del negocio. A veces se les suele denominar Business Objects, y también usar el sufijo BO para su nombre: GestorContratosBO.

Comprobación de que responde cada vez una instancia

```
...
public String saludo(String nombre) {
    int random = (int) (Math.random() * misSaludos.length);
    String saludo = nombre + ", " + misSaludos[random] + " [" + this.toString() + "]";
    return saludo;
}
...
```

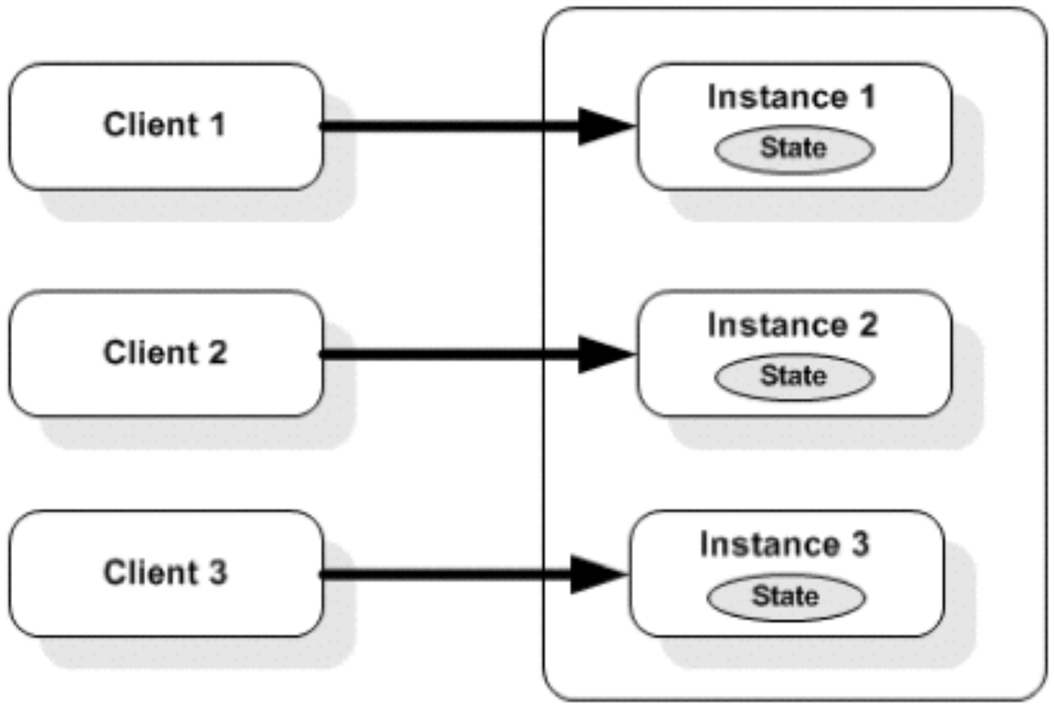
Añadimos al saludo la referencia de la instancia

```
...
out.println("<h1>Stateless</h1>");
out.println("<ul>");
out.println("<li>" + saludoServicio.saludo(nombre) + "</li>");
out.println("<li>" + saludoServicio.saludo(nombre) + "</li>");
out.println("</ul>");
...
```

Invocamos dos veces, nos responderán distintas instancias

Beans de sesión con estado

- El contenedor asigna un bean a cada cliente y mantiene esta asignación mientras que vive el cliente
- Esto permite que el bean tenga variables de instancia que almacenan un estado cambiante
- El cliente invoca a sus métodos y modifica ese estado



Ejemplo

Anotación @Stateful

Estado asociado a la instancia

Obtenemos por inyección el bean que devuelve un saludo

```
@Stateful
public class SaludoConEstadoServicio {
    ArrayList<String> saludos = new ArrayList<String>();

    @EJB
    SaludoServicio saludoServicio;

    public String saludo(String nombre) {
        String saludo = saludoServicio.saludo(nombre);
        saludos.add(saludo);
        return saludo;
    }

    public ArrayList<String> saludos() {
        return saludos;
    }
}
```

El método de negocio guarda el saludo en el array local

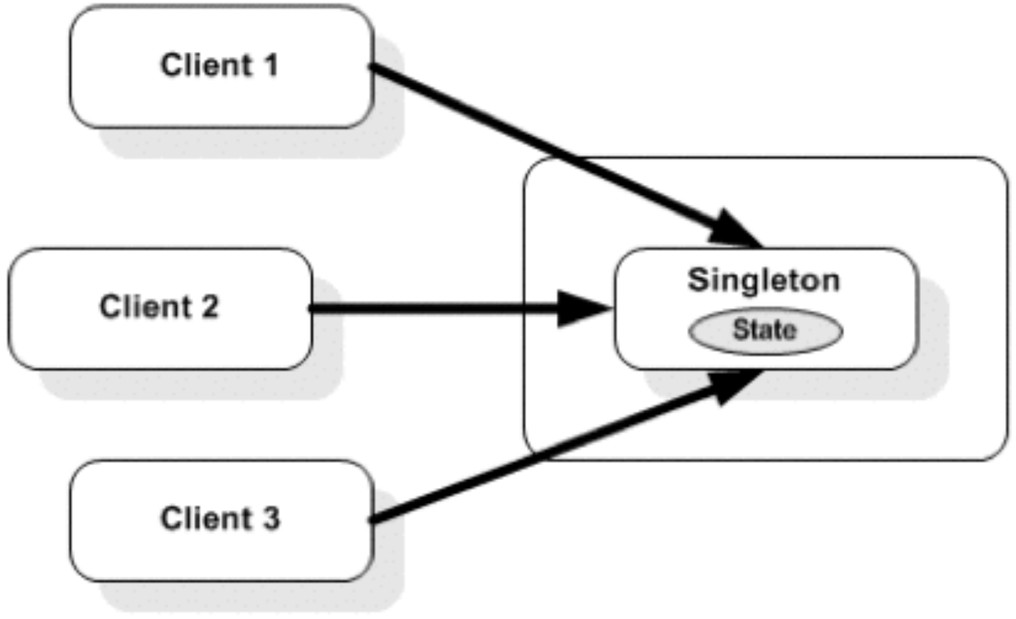
Otro método de negocio devuelve los saludos almacenados hasta el momento

Beans de sesión singleton

- Un único bean para todo el tiempo de vida de la aplicación
- Múltiples clientes van a compartir el mismo bean

Anotación
@Singleton

```
@Startup
@Singleton
public class MySingleton {
    //...
}
```



Ejemplo

Se inicializa al arrancar

Anotación @Singleton

Acceso secuencial de clientes concurrentes

Acceso paralelo de clientes concurrentes

```
@Startup
@Singleton
public class SaludoSingletonServicio {
    ArrayList<String> saludos = new ArrayList<String>();

    @EJB
    SaludoServicio saludoServicio;

    @Lock(LockType.WRITE)
    public String saludo(String nombre) {
        String saludo = saludoServicio.saludo(nombre);
        saludos.add(saludo);
        return saludo;
    }

    @Lock(LockType.READ)
    public ArrayList<String> saludos() {
        return saludos;
    }
}
```

Acceso al bean mediante inyección de dependencias

- En las variables de instancia de objetos gestionados: servlets, páginas JSP, CDIs, otros enterprise beans
- Cuando el contenedor crea el objeto gestionado realiza la inyección
- Cuidado con los servlets: su creación sólo se hace una vez y todos los hilos comparten las variables de instancia. Sólo hay que inyectar beans de sesión sin estado.

```
@WebServlet(name="holamundo", urlPatterns="/holamundo")
public class HolaMundoServlet extends HttpServlet {

    @EJB
    SaludoServicio saludoServicio;

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException, IOException {
        response.setContentType(“text/html");
      ...
    }
}
```

Acceso al bean mediante su nombre JNDI

- Cuando se despliega la aplicación en el servidor, éste proporciona un nombre estándar a cada enterprise bean
- Tres nombres con prefijos distintos para acceder desde distintos ámbitos
  - Acceso remoto (desde otro servidor de aplicaciones):
    `java:global[/nombre aplicación]/nombre módulo/nombre enterprise bean`
  - Acceso local (mismo servidor de aplicaciones) desde otra aplicación (otro WAR):
    `java:app[/nombre módulo]/nombre enterprise bean`
  - Acceso local desde el mismo WAR:
    `java:module/nombre enterprise bean`
- El servidor de aplicaciones muestra por la consola los nombres asignados
- Para obtener la referencia al bean se llama al método `lookup()` de un objeto `InitialContext`

Ejemplo obtención referencia con JNDI

- Podemos utilizar la localización por JNDI para acceder a un bean con estado distinto desde cada ejecución de un servlet:

```
@WebServlet(name="holamundoestado", urlPatterns="/holamundoestado")
public class HolaMundoConEstadoServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                             throws ServletException, IOException {

        response.setContentType("text/html");
        SaludoConEstadoServicio saludoServicio = (SaludoConEstadoServicio)
              new InitialContext().lookup(“java:module/SaludoConEstadoServicio");
        ...
    }
}
```

Llamada a lookup con
el nombre JNDI del bean

Pruebas de enterprise beans con Arquillian

- El framework de JBoss Arquillian permite realizar testing de componentes desplegándolos en el servidor de aplicaciones y lanzando los tests, que se ejecutan como si se invocaran dentro del servidor.
- Dos configuraciones de testing posibles:
  - Servidor de aplicaciones gestionado: Arquillian se encarga de poner en marcha el servidor de aplicaciones, desplegar los componentes y los tests, lanzarlos y cerrar el servidor de aplicaciones.
  - Servidor de aplicaciones remoto: servidor de aplicaciones en marcha y Arquillian se comunica con él para desplegar los componentes y los tests. Esta forma es más eficiente.
- La configuración de testing usada depende de las dependencias definidas en Maven, los tests son iguales para ambas configuraciones

Configuración de Maven

- Sólo hay que añadir dos dependencias para poder ejecutar los tests en una configuración de WildFly remota

```
<dependency>
    <groupId>org.jboss.arquillian.junit</groupId>
    <artifactId>arquillian-junit-container</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.wildfly</groupId>
    <artifactId>wildfly-arquillian-container-remote</artifactId>
    <version>8.1.0.Final</version>
    <scope>test</scope>
</dependency>
```

Ejemplo

Anotación
@RunWith

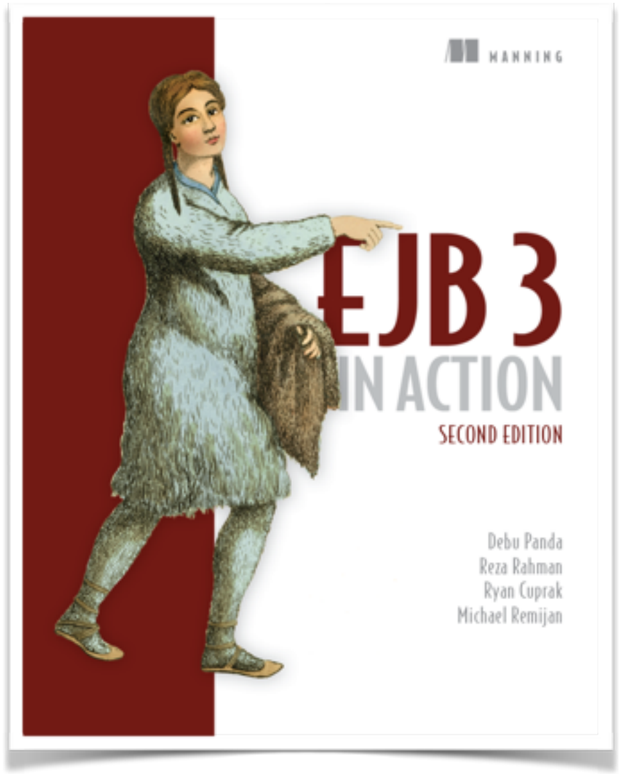
Se despliegan
las clases
a probar

En el test se
invoca al bean
igual que en el código
real, no hay que usar
mocks, ni stubs

```java
import static org.junit.Assert.assertTrue;

@RunWith(Arquillian.class)
public class SaludoServicioTest {

    @EJB
    private SaludoServicio saludoServicio;

    @Deployment
    public static Archive<?> deployment() {
        return ShrinkWrap.create(JavaArchive.class)
                .addClass(SaludoServicio.class);
    }

    @Test
    public void deberiaDevolverUnoDeLosSaludos() {
        String[] misSaludos = {"Hola, que tal?",
                "Cuanto tiempo sin verte", "Que te cuentas?",
                "Me alegro de volver a verte"};
        String nombre = "Pepito";
        String saludo = saludoServicio.saludo(nombre);
        assertTrue(saludo.startsWith(nombre + ", " + misSaludos[0]) ||
                   saludo.startsWith(nombre + ", " + misSaludos[1]) ||
                   saludo.startsWith(nombre + ", " + misSaludos[2]) ||
                   saludo.startsWith(nombre + ", " + misSaludos[3]));
    }
}
```

Para saber más

- Debu Panda y otros:
  EJB 3 in Action, second edition
  Ed. Manning, 2013



¿Preguntas?