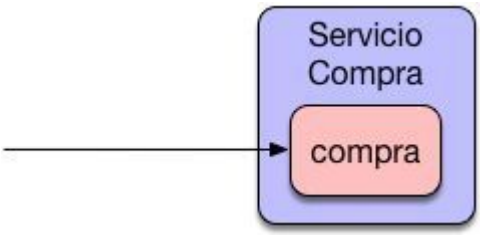
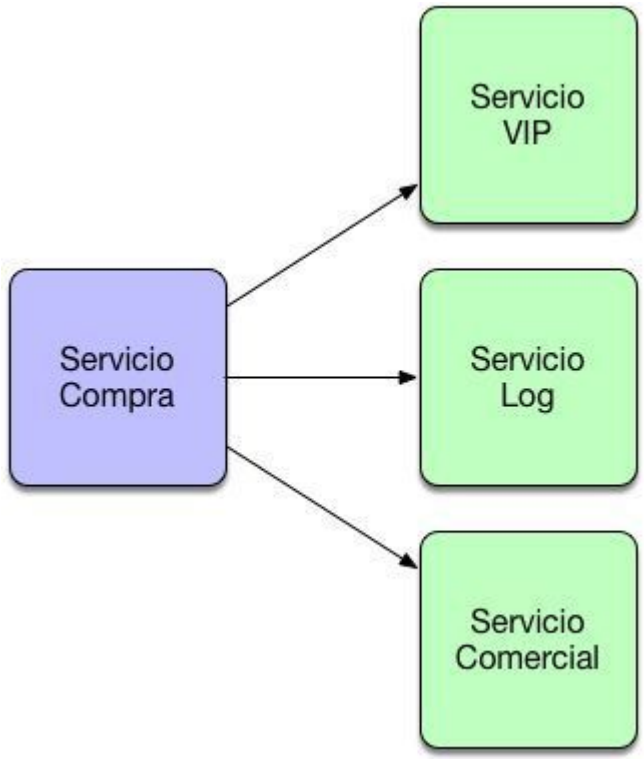
El concepto de EJB Event pertenece a Java EE 6 . A partir de esta versión de la plataforma tenemos la posibilidad de generar eventos a nivel de EJBs. Esto es algo que en un principio parece extraño ya que estamos más acostumbrados a la gestión de eventos en el mundo de JavaScript y la web. ¿Para qué sirve la generación de eventos a nivel de EJBS?. Vamos a ver un ejemplo que nos ayude a entender la casuística. Supongamos que tenemos un Stateless Session Bean que se encarga de confirmar una compra VIP (ServicioCompra).



Estas compras VIP son muy importantes para nuestro negocio ya que identifican a nuestros mejores clientes. Por lo tanto cada vez que realizamos una compra VIP se realizan varias operaciones asociadas. En este caso se realiza un log especial, se envía un mensaje al comercial y envía al propio cliente un mensaje de bienvenido al servicio.



Vamos a ver el código Java utilizando 4 EJBS:

```
package com.arquitecturajava;

import javax.ejb.Stateless;
import javax.inject.Inject;

@Stateless
public class ServicioCompra {

	@Inject
	ServicioLog log;

	@Inject
	ServicioVip vip;

	@Inject
	ServicioAvisos avisos;

	public void confirmarCompra(Compra c) {

		System.out.println("compra guardada");
		if (c.getImporte()&amp;gt;1000) {

		log.VIP(c);
		avisos.avisarComercial();
		vip.mensajeVip();
		}
	}
```

```
}
```

```
package com.arquitecturajava;

import javax.ejb.Stateless;

@Stateless
public class ServicioLog {

        public void VIP (Compra c) {

                System.out.println("COMPRA VIP"+ c.getImporte());
        }
}
```

```
package com.arquitecturajava;

import javax.ejb.Stateless;

@Stateless
public class ServicioVip {

        public void mensajeVip() {

                System.out.println("bienvenido al servicio vip");
        }
```

```
}
```

```
package com.arquitecturajava;

import javax.ejb.Stateless;

@Stateless
public class ServicioAvisos {

        public void avisarComercial() {

                System.out.println("compra importante avisa a un
comercial");
        }
}
```

Todo funciona de una forma muy correcta y nuestro ServicioCompra invoca el resto de EJBS .Invocamos al EJB de ServicioCompras desde un Servlet.

Veamos su código:

```
package com.arquitecturajava;

import java.io.IOException;
import java.io.PrintWriter;
```

```
import javax.inject.Inject;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/miServlet")
public class MiServlet extends HttpServlet {

        @Inject
        ServicioCompra sc;

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

                PrintWriter pw = response.getWriter();
                sc.confirmarCompra(new Compra("televisor", 1200));
                pw.println("compra vip realizada");

        }

}
```

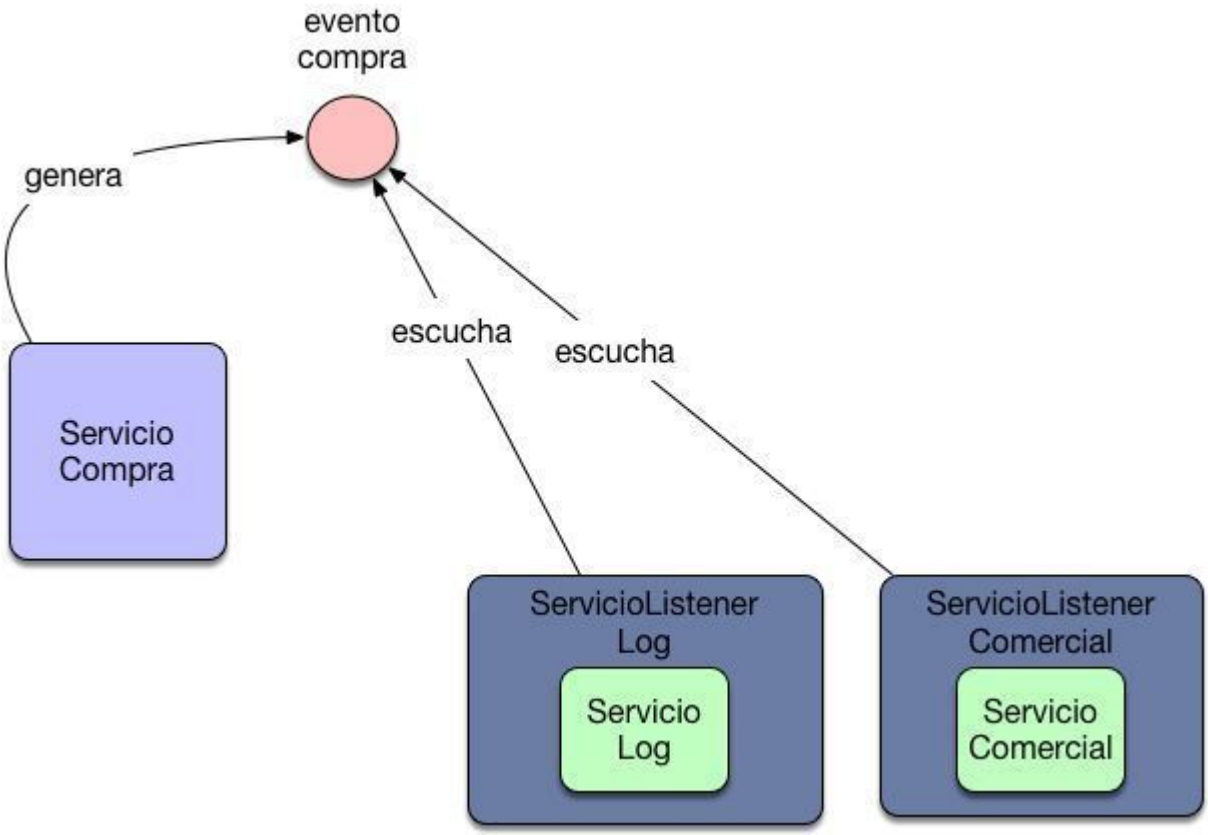
El resultado que nos saldrá en el navegador es:

La consola imprimirá los mensajes de ejecución de cada EJB.

¿Es mejorable el código que hemos construido? . El problema es que para ser un servicio tan importante de la aplicación y que puede variar mucho el EJB de Servicio compra tiene un acoplamiento muy fuerte y muy directo con los otros 3 servicios. En un futuro pueden ser más o incluso pueden variar los que ahora mismo funcionan. Esto hará que nuestro código sea muy frágil y de miedo modificar esta parte. ¿Cómo podemos enfocar de otra forma?

EJB Event , desacoplando servicios

Una buena solución es construir un EJB Event de tal forma que cuando la compra se realice y sobrepase los mil euros sea lanzado . Para luego poder utilizar servicios de EJB que escuchen este evento.



Vamos a verlo en código, el primer paso es generar el evento que contenga una compra:

```
package com.arquitecturajava;

public class EventoCompraVip {

	private Compra compra;

	public Compra getCompra() {
		return compra;
	}

	public void setCompra(Compra compra) {
```

```
                this.compra = compra;
        }

        public EventoCompraVip(Compra compra) {
                super();
                this.compra = compra;
        }

}
```

El siguiente paso es lanzarlo desde un EJB :

```
package com.arquitecturajava;

import javax.enterprise.event.Event;
import javax.inject.Inject;

public class ServicioCompra {

        @Inject
        Event&amp;lt;EventoCompraVip&amp;gt; misEventos;

        public void confirmarCompra(Compra c) {

                System.out.println("compra guardada");
                if (c.getImporte()&amp;gt;1000) {

                        misEventos.fire(new EventoCompraVip(c));
```

```
                }
        }

}
```

Por último nos queda diseñar uno o mas listener que se encarguen de gestionar el evento:

```
package com.arquitecturajava.listener;

import javax.ejb.Stateless;
import javax.enterprise.event.Observes;
import javax.inject.Inject;

import com.arquitecturajava.EventoCompraVip;
import com.arquitecturajava.ServicioAvisos;
import com.arquitecturajava.ServicioLog;

@Stateless
public class ServicioAvisosListener {

        @Inject
        ServicioAvisos servicio;

        public void escuchaCompra(@Observes EventoCompraVip evento) {

                servicio.avisarComercial();

        }
```

```
}
```

```
package com.arquitecturajava.listener;

import javax.ejb.Stateless;
import javax.enterprise.event.Observes;
import javax.inject.Inject;

import com.arquitecturajava.EventoCompraVip;
import com.arquitecturajava.ServicioLog;

@Stateless
public class ServicioListenerLog {

        @Inject
        ServicioLog servicio;

        public void escuchaCompra(@Observes EventoCompraVip evento) {

                System.out.println("COMPRA VIP"+
evento.getCompra().getImporte());

        }

}
```

El resultado cuando ejecutemos el servlet , mostrará la ejecución de los listeners de ese EJB

Event, hemos desacoplados los EJB:

```
INFORMACION: Server startup in 694 ms
compra guardada
compra importante avisa a un  comercial
COMPRA VIP1200.0
```

El uso de los EJB Event es muy práctico cuando queremos desacoplar elementos y añadir flexibilidad:

Otros artículos relacionados:

1. Usando un EJB Async
2. EJB Singleton
3. EJB (V) EJBs Remotos
4. EJB