Tema Componentes: Programación de componentes

Componente (.jar): Unidad de software independiente de la aplicación, posee un conjunto de interface y requisitos (independiente a la aplicación y funcional para cualquier aplicación)

Características de un componente:

- Independiente de la plataforma
- Identificable: Su nombre debe indicar su funcionalidad
- Autocontenido: No puede requerir otros componentes para llevar a cabo su funcionalidad
- Remplazable por otro componente
- Acceso solamente a través de su interfaz: No se puede acceder al código fuente
- Sus servicios no varían
- Bien documentado
- Genérico
- Reutilizado dinámicamente: Cargado en tiempo de ejecución
- Se distribuye como un paquete

Tecnologías de empaquetamiento de componentes:

- Modelo de componentes: Reglas de diseño que deben obedecer los componentes, sus interfaces y la iteración de los mismos
- Plataforma de componentes: Infraestructura de software requerida para la ejecución de aplicaciones basadas en componentes, basados en un determinado modelo Ejemplos:
  - .NET de Microsoft
  - JavaBeans (JB): Diseñado para ser ejecutado en un cliente
  - Enterprise JavaBeans (EJB): Diseñado para ser ejecutado en un servidor

Ventajas e inconvenientes:

Ventajas	Inconvenientes
Reutilización de software	Solo existen algunos campos como las GUIs y no siempre se pueden encontrar los componentes adecuados
Disminución de la complejidad de software	
Los errores son más fáciles de detectar	
Incrementa la calidad del software, pueden ser reconstruidos para ser mejorados	

JavaBeans:

Además de cumplir las características de los componentes, tiene que cumplir las siguientes:

- Introspección: Mecanismo mediante el que el JB proporciona información sobre sus: propiedades, métodos y eventos.
  Existen dos formas: patrones de nombrado y la clase BeanInformation (proporciona características)
- Manejo de eventos: Método de comunicación entre JB
- Propiedades: Determinan la apariencia y comportamiento de un Bean
- Persistencia (Serializable):
- Personalización: Permite una alteración de la apariencia y conducta durante su desarrollo

Propiedades y atributos:

- *Propiedades simples: Representan un único valor de tipo primitivo o referenciado (objeto)*
- *Propiedades indexadas: Representa una colección, debe poseer getters y setters (ej: array)*
- *Propiedades ligadas*
- *Propiedades restringidas*

POJOS

JAR

Propiedades ligadas.

Propiedades asociadas a eventos, pertenecen a la clase PropertyChangeSupport permite generar eventos no visuales tiene diferentes métodos que pueden realizar cambios en propiedades, estés cambios los realiza el **BeanOyente** sobre el **BeanFuente**:

- En el **BeanFuente** (implementa Serializable):

```
45    public void addPropertyChangeListener(PropertyChangeListener listener) {
46        propertySupport.addPropertyChangeListener(listener);
47    }
48
49    public void removePropertyChangeListener(PropertyChangeListener listener) {
50        propertySupport.removePropertyChangeListener(listener);
51    }
```

Además de un método set que implemente firePropertyChange()

```
22    public void setStock(int stockNue) {
23
24        int stockAnt = stock;
25        stock = stockNue;
26
27        if (stock < getStockMinimo()) { //realizar pedido
28
29            propertySupport.firePropertyChange("pedidoSi", stockAnt, stock);
30        } else {
31
32            propertySupport.firePropertyChange("pedidoNo", stockAnt, stock);
33        }
34    }
```

- En el **BeanOyente** (implementa Serializable, PropertyChangeListener):

```
71      @Override
72      public void propertyChange(PropertyChangeEvent evt) {
73
74          if (evt.getPropertyName().compareToIgnoreCase("peidoSi") == 0) {
75              System.out.println("\n Producto " + producto.getDescripcion() + ": Stock anterior: "
76                      + evt.getOldValue() + " - Stock actual: " + evt.getNewValue() +
77                      "\n Es necesario realizar un pedido de: " + producto.getDescripcion() +
78                      "\n El stock actual esta por debajo del stock minimo recomendado \n");
79              setPedir(true);
80          } else {
81              System.out.println("\n Producto " + producto.getDescripcion() + ": Stock anterior: "
82                      + evt.getOldValue() + " - Stock actual: " + evt.getNewValue() +
83                      "\n No es necesario realizar un pedido de: " + producto.getDescripcion() +
84                      "\n El stock actual esta por encima del stock minimo recomendado \n");
85              setPedir(false);
86          }
87      }
```

Propiedades restringidas:

Similares a las ligadas, pero en este caso los objetos el **BeanOyente** puede dejar de escuchar los eventos, para ello deben implementarse los siguientes métodos:

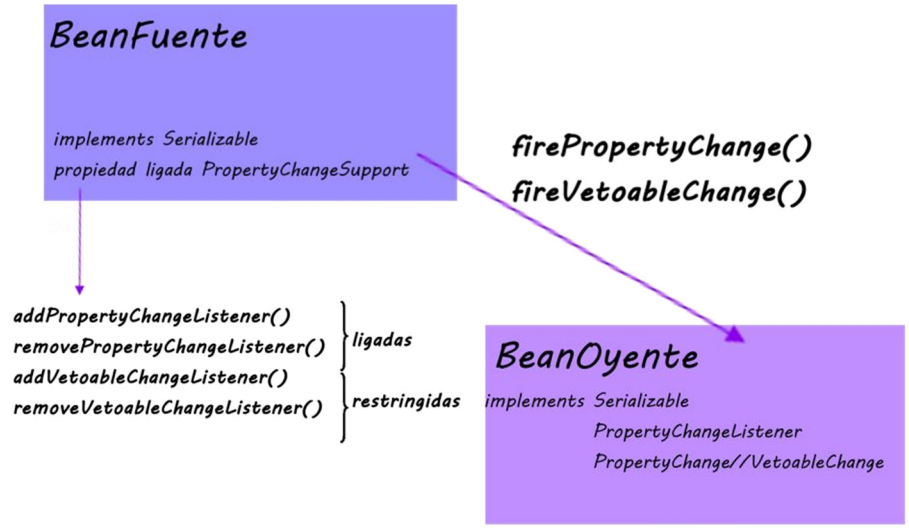
- En el **BeanFuente** (implementa Serializable):

```
2       public void addVetoableChangeListener(VetoableChangeListener listener) {
3           soporteVeto.addVetoableChangeListener(listener);
4       }
5
6       public removeVetoableChangeListener(VetoableChangeListener listener) {
7           soporteVeto.removeVetoableChangeListener(listener);
8       }
```

- En el **BeanOyente** (implementa Serializable, VetoableChangeListener):

```
2       public void vetoableChange(PropertyChangeEvent evt) throws PropertyVetoException {
3           //comprobacion de las condiciones
4           //se lanza excepcion si no se aprueba el cambio
5           throw new PropertyVetoException ("mensaje", evt);
6       }
```

Resumen: propiedades:



Eventos:

Los Beans utilizan los eventos para comunicarse con otros Beans

- Para lanzar eventos PropertyChangeEvent a los BeansOyentes:
  - firePropertyChange(nombrePropiedad, valorAntiguo, valorNuevo) Para lanzar eventos PropertyVetoEvent a los BeansOyentes:
  - fireVetoableChange(nombrePropiedad, valorAntiguo, valorNuevo)
  - Captura excepciones PropertyVetoException

Persistencia del componente:

Mediante el mecanismo de persistencia un Bean es capaz de almacenar su estado en un momento determinado y recuperarlo posteriormente, para ello usamos la serialización teniendo en cuenta que se serializa todo menos los campos static y transient, este último es usado para indicar que no se debe seriar un campo