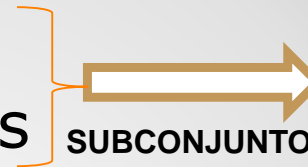


**UD3**

# Creación de componentes visuales

- Clase creada para ser **reutilizada**

- Estado - propiedades
- Comportamiento - eventos



INTERFAZ DEL  
COMPONENTE

- Requisitos:

- **Persitencia**
- Permitir **modificarse**
- **Introspección**
- Gestión de **eventos**

- **Desarrollo basado en componentes**

- Más sencillo y rápido
- Menos errores

## ¿Qué es un componente?

- Un **JavaBean** es un componente software reusable que puede ser manipulado visualmente mediante una herramienta gráfica
- **Persistencia**
  - implements *Serializable*
    - Necesario **constructor** sin parámetros
- Una vez implementado... se puede añadir a la **paleta** en una **categoría**
  - Ej. Componentes personalizados



# JavaBeans

- Las propiedades de un componente pueden examinarse y **modificarse**:

- Métodos **getter**

```
public <TipoPropiedad> get<NombrePropiedad>( )
```

- Métodos **setter**

- Si no se incluyen es *sólo lectura*

```
public void set<NombrePropiedad>(<TipoPropiedad> valor)
```

- Tiene que ser *serializables*
- Cambios a través del IDE: **Editor de propiedades**

## Propiedades de un componente

- Cada propiedad asociada a un editor:
  - Clases básicas, Font y Color lo da la herramienta
  - Otros casos:
    - Crear panel personalizado
    - Crear clase de vínculo entre la propiedad y el panel:

```
public <Propiedad>Editor implements PropertyEditorSupport
```

- *supportsCustomEditor*: devolvería **true**
- *getCustomEditor*: devuelve el panel del editor personalizado
- *getValue*: devuelve el valor que genera el panel con la información introducida por el usuario
- *getJavaInitializationString*: hay que componer a mano el valor de inicialización

## Editor de propiedades

- **Introspección**: Permite arrastrar y soltar un componente en la zona de diseño y determinar **dinámicamente** qué propiedades y eventos del componente están disponibles
  - Para conseguirlo se utiliza la **reflexión**
- **BeanInfo**
  - Clase asociada con información del componente
  - Se crea mediante clic derecho en el componente, **Editor BeanInfo**
    - Modificar **Property Editor Class** (ruta completa de la clase heredera de *PropertyEditorSupport*)

## Introspección y reflexión

- Para que el componente pueda reconocer el evento y responder ante el...

- Definir **interfaz** que represente el **listener** del evento
  - Debe tener un método para procesar el evento

```
public interface ArrastreListener {  
    public void arrastre();  
}
```

- En el componente, definir dos operaciones, para añadir y eliminar oyentes

```
public void addArrastreListener(ArrastreListener l) public  
void removeArrastreListener(<ArrastreListener l)
```

- El componente establece cuándo hay que llamar al manejador:

```
if (Math.abs(puntoPresion.x-posicionActual.x)>50){  
    arrastreListener.arrastre();  
}
```

- Cuando se usa el componente es cuando se define qué hacer si surge el evento

```
panel.addArrastreListener(new ArrastreListener(){  
    public void arrastre() {  
        System.out.println("Has arrastrado");  
    }  
});
```

# Eventos

- Distribución en **paquete jar**
  - Componente
  - Objetos **BeanInfo**
  - Objetos editor de propiedades
  - Clases o recursos que requiera el componente
  - Fichero de manifiesto (.mf) con la descripción
- Creación del jar
  - Netbeans: **Limpiar y construir**
  - Comando **jar cfm**

**Por último... el empaquetado**