Alumno: Kevin Zamora Amela
Tarea: Tarea 6 (AD07)
Asignatura: Acceso a Datos

Programación Orientada a Componentes y JavaBeans

1. Concepto de Componente Software

Un componente software es un elemento modular y autónomo que puede integrarse fácilmente en sistemas más grandes (javadeploy.com). Está diseñado como un bloque de construcción fundamental que facilita la creación de aplicaciones complejas mediante la combinación de elementos reutilizables.

2. Ventajas e Inconvenientes

Ventajas	Inconvenientes
Reutilización de código	Mayor complejidad inicial
Facilidad de mantenimiento	Necesidad de documentación detallada
Integración flexible	Posible sobrecarga en sistemas pequeños
Desarrollo paralelo	Requiere planificación cuidadosa
Pruebas más simples	Curva de aprendizaje inicial

3. Herramientas para Desarrollo de Componentes

- Frameworks de componentes (JavaBeans, Spring)
- IDEs especializados (NetBeans, Eclipse)
- Herramientas de empaquetado (Maven, Gradle)
- Entornos de desarrollo integrados

4. JavaBean: Concepto y Requisitos

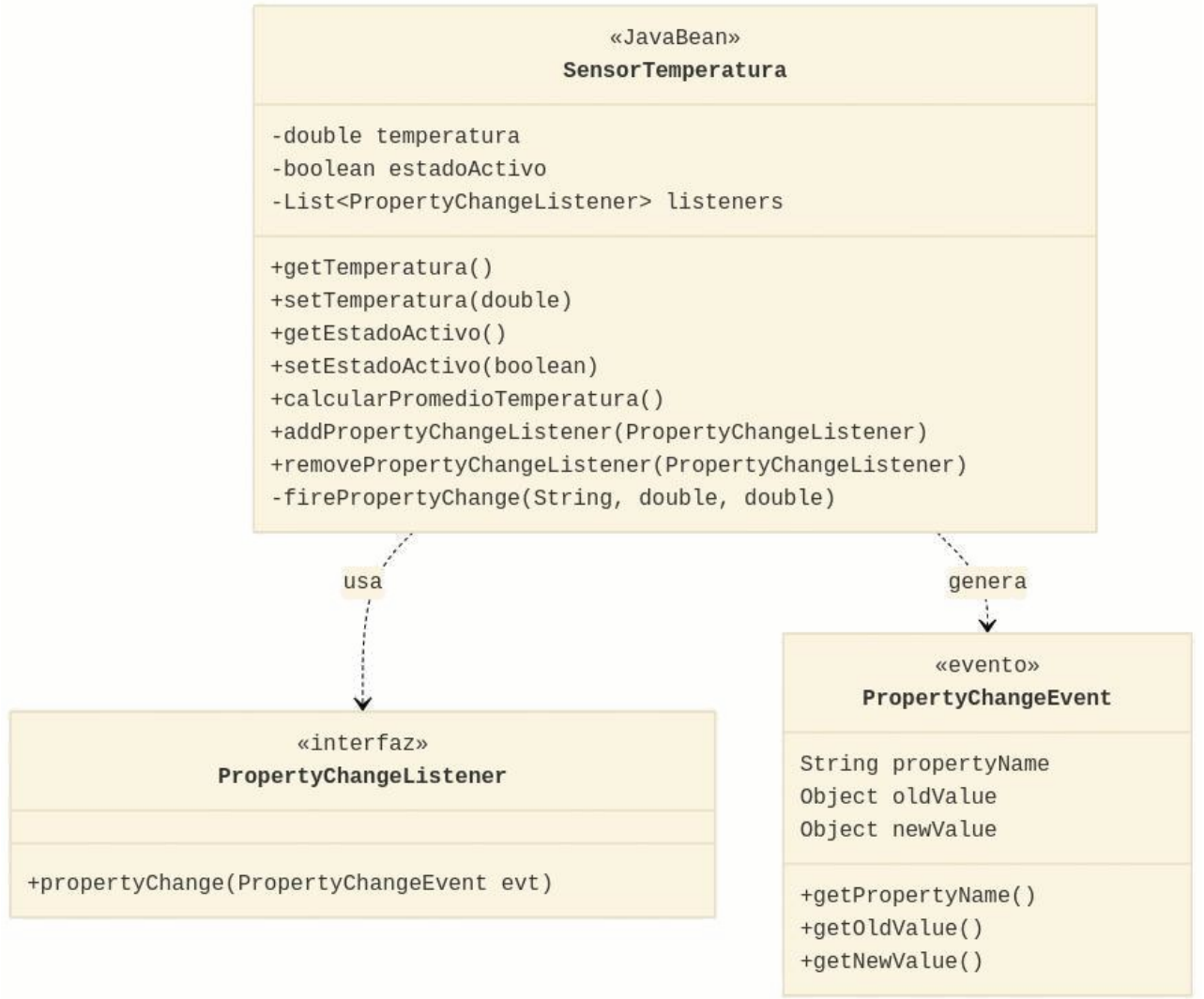
Un JavaBean es un componente software que sigue ciertas convenciones específicas (stackoverflow.com):

- Implementación de la interfaz Serializable
- Constructor público sin argumentos
- Propiedades privadas con getters y setters públicos
- Encapsulación adecuada de datos

5. Componente JavaBean: SensorTemperatura

Para cumplir con los requisitos del ejercicio, hemos desarrollado un componente JavaBean que simula un sensor de temperatura. Este componente será funcional y educativo, permitiendo entender todos los conceptos clave de los JavaBeans.

Primero, veamos la estructura del componente:



Explicación del Diagrama

El diagrama muestra la estructura completa de nuestro componente JavaBean SensorTemperatura:

- Las flechas punteadas (..>) indican dependencias: SensorTemperatura usa PropertyChangeListener y genera eventos PropertyChangeEvent
- PropertyChangeListener es una interfaz que define el contrato para los oyentes de eventos
- PropertyChangeEvent es una clase que encapsula la información del evento cuando cambian las propiedades
- SensorTemperatura implementa todas las características requeridas de un JavaBean, incluyendo propiedades privadas y métodos públicos

Implementación del Componente (Código)

```
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class SensorTemperatura implements Serializable {
    private static final long serialVersionUID = 1L;

    // Propiedades privadas
    private double temperatura;
    private boolean estadoActivo;

    // Lista de listeners para eventos
    private List<PropertyChangeListener> listeners = new ArrayList<>();

    // Constructor sin argumentos
    public SensorTemperatura() {
        this.temperatura = 0.0;
        this.estadoActivo = false;
    }

    // Getters y setters
    public double getTemperatura() {
        return temperatura;
    }

    public void setTemperatura(double temperatura) {
        double oldValue = this.temperatura;
        this.temperatura = temperatura;
        firePropertyChange("temperatura", oldValue, temperatura);
    }

    public boolean getEstadoActivo() {
        return estadoActivo;
    }

    public void setEstadoActivo(boolean estadoActivo) {
        boolean oldValue = this.estadoActivo;
        this.estadoActivo = estadoActivo;
        firePropertyChange("estadoActivo", oldValue, estadoActivo);
    }

    // Método de comportamiento
    public double calcularPromedioTemperatura(int numMuestras) {
        if (!estadoActivo) {
            throw new IllegalStateException("El sensor debe estar activo");
        }
        // Simulación del cálculo promedio
        return temperatura * numMuestras / numMuestras;
    }

    // Gestión de eventos
    public void addPropertyChangeListener(PropertyChangeListener listener) {
        listeners.add(listener);
    }

    public void removePropertyChangeListener(PropertyChangeListener listener) {
        listeners.remove(listener);
    }

    private void firePropertyChange(String propertyName, double oldValue, double
newValue) {
```

```
        PropertyChangeEvent evt = new PropertyChangeEvent(this, propertyName,
oldValue, newValue);
        for (PropertyChangeListener listener : listeners) {
            listener.propertyChange(evt);
        }
    }
}
```