

Módulo Profesional 06:
Acceso a datos

Actividad UF4

CICLO FORMATIVO DE GRADO SUPERIOR EN

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

MODALIDAD ONLINE

Enrique Verea



Componentes de acceso a datos

Objetivos

Programar con componentes de acceso a datos, usando los eventos.

Competencias asociadas:

- Programación con componentes (NetBeans)
- Programación usando los eventos

Metodología

Entrega

Preparación individual

12 diciembre 2023, en PDF + ZIP

Dedicación estimada

Documentos de referencia

8 horas

Resultados de aprendizaje

- NetBeans
- Programación con eventos

Criterios de evaluación

- Programar con componentes
- Programar con eventos
- Incluye comentarios en tu código para facilitar su mantenimiento en un futuro hipotético

Desarrollo de la actividad

¡IMPORTANTE!

Para cada uno de los ejercicios, se deberá:

- explicar el ejercicio para que quede claro qué está haciendo y porqué (escribirlo en este documento, que luego guardaréis como pdf)
- documentar con **capturas de pantalla** los pasos más significativos del funcionamiento del programa, intercalándolo con el punto anterior
- documentar los **resultados** obtenidos al finalizar la ejecución del programa
- incluir también **comentarios en el código** para explicar los pasos más importantes
- copiar el **código completo** (ponerlo al final de este documento, en un apartado llamado **Anexo**, indicando el número de ejercicio al que pertenece)

Para **presentar** la actividad:

- se deberán subir, en el único intento habilitado disponible, los **2 archivos**: 1 pdf y 1 zip
 - o el pdf incluirá este documento con toda la resolución de los ejercicios (con lo comentado anteriormente)
 - o el zip incluirá sólo los archivos zip (con el código de cada uno de los ejercicios). Es decir, el zip englobará a los otros zips (1 por actividad)
- Por favor, no pongáis el pdf dentro del zip 😊

JAVABEANS

Ejercicio1.

¿Qué es un JavaBean? ¿Qué características debe incluir esta clase de Java para que se considere un JavaBean? Explicálo con tus palabras.

Es un modelo de componente que consiste en una clase Java que encapsula otras clases para crear un componente reutilizable, interoperable y muy fácil de utilizar y mantener. Puede ser manipulado directamente por diferentes frameworks sin necesidad de configuraciones adicionales y suele ser utilizado para representar y compartir datos del mundo real.

Para que una clase Java sea considerada un componente JavaBean debe seguir la siguiente convención:

- Debe tener un constructor sin argumentos
- Sus atributos deben ser privados
- Debe exponer métodos getter y setter para todas sus propiedades, siguiendo el estándar de nomenclatura de Java para estos métodos: *getAtributo*, *setAtributo*
- Debe ser serializable, es decir, implementar la interfaz *Serializable*. (para ser serializable, todos los miembros de la clase también deben ser serializables, o estar marcados como *'transitorios'*)

Ejercicio2.

- Es flexible y fácil de implementar. El programa no tiene ningún orden específico que tenga que ser mantenido por el programador si no que se ejecuta dependiendo de los eventos que vayan generando.
- Mantiene la separación de intereses entre distintas clases, ya que no es necesario que las clases involucradas tengan conocimientos una de otra, solo basta con registrar una clase como observador de los eventos generados por otra.
- Mantiene la flexibilidad y modularidad del código y lo hace resistente al cambio. Para implementar nuevos requerimientos en la lógica del programa basta con generar nuevos eventos y registrar observadores para dichos eventos. Esto hace que no haya

necesidad de alterar el código existente, sino más bien extenderlo con el código necesario para generar y escuchar los nuevos eventos.

- Compagina muy bien con programas que implementan una interfaz de usuario gráfica, ya que el código puede responder a los eventos generados por el usuario al interactuar con la interfaz.

Ejercicio3.

Genera un programa de control de análisis clínicos. Se trata de guardar los datos de análisis de una persona (Nombre, nivel de hierro, urea).

En el momento que una persona tenga una variación de hierro > 10, significa que algo pasa, por lo que se le debe hacer otro análisis.

Implementación

El ejercicio utiliza las clases Paciente, GeneraAnálisis y Hospital. Para solicitar entrada de datos al usuario utiliza la clase InputUsuario (ver la clase en la sección Anexos).

Paciente

La clase Paciente es un JavaBean con las siguientes propiedades:

- nombre : String
- apellido : String
- edad : int
- telefono : String
- ultimoHierro : int
- ultimaUrea : int

La clase expone el método `addPropertyChangeListener(PropertyChangeListener)` para registrar observadores a los eventos que genera al recibir cambios en sus propiedades. En esta implementación, la clase Paciente solo genera eventos cuando hay cambios en la propiedad 'ultimoHierro'.

GeneraAnálisis

La clase GeneraAnálisis es un JavaBean con las siguientes propiedades:

- numAnálisis : int
- paciente : Paciente
- fecha : Date
- analisisPendiente : int

La clase implementa la *interfaz* `PropertyChangeListener` y observa cambios en la propiedad 'ultimoHierro' de la clase Paciente. GeneraAnálisis clase se añade a sí misma como observador a la clase Paciente en su método `setPaciente(Paciente)`

```
public void setPaciente(Paciente paciente) {  
    this.paciente = paciente;  
    paciente.addPropertyChangeListener(this);  
}
```

Si los valores de hierro recibidos en los eventos de cambio presentan una diferencia entre ellos mayor a 10, el valor de la propiedad analisisPendiente es aumentado a 1.

```
@Override
public void propertyChange(PropertyChangeEvent evt) {
    if (evt.getPropertyName().equalsIgnoreCase("ultimoHierro")) {

        int variacionHierro = diferenciaEntreValores(evt.getOldValue(), evt.getNewValue());

        if (variacionHierro > 10)
            analisisPendiente++;
    }
}
```

GeneraAnálisis también ofrece un método para generar nuevos análisis a partir de otro y así mantener una numeración constante

```
public GeneraAnálisis generarNuevoAnálisis(GeneraAnálisis antiguo, Date fecha) {
    GeneraAnálisis nuevoAnálisis = new GeneraAnálisis(
        antiguo.getNumAnálisis() + 1,
        fecha);

    nuevoAnálisis.setPaciente(antiguo.getPaciente());
    return nuevoAnálisis;
}
```

Hospital

La clase hospital es la clase Main del programa. Se encarga de instanciar las clases Paciente, GeneraAnálisis y Hospital, y la clase InputUsuario para solicitar entrada de datos al usuario.

El programa corre un bucle que itera mientras existan análisis pendientes al Paciente.

En el bucle se solicitan valores de hierro y úrea al usuario y se registran en la instancia de Paciente.

```
int valorHierro = input.solicitarEntero("Introducir valor de hierro: ");
paciente.setUltimoHierro(valorHierro);

int valorUrea = input.solicitarEntero("Introducir valor de úrea: ");
paciente.setUltimaUrea(valorUrea);
```

Luego se determina si es necesario generar un nuevo análisis, evaluando el valor de la propiedad analisisPendiente de la instancia de GeneraAnálisis: si el valor es mayor a '0', un nuevo análisis es necesario.

```

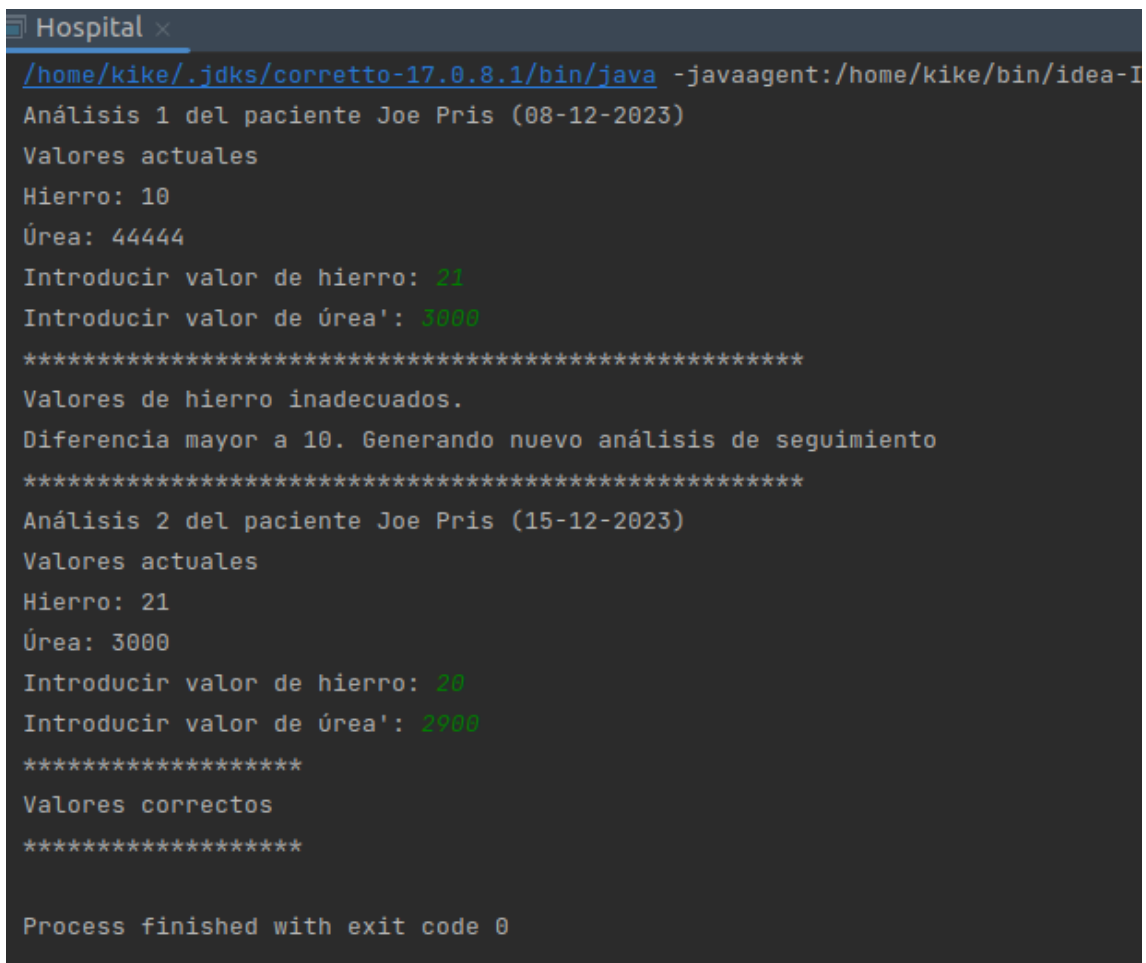
if (analisis.getAnalisisPendiente() == 0) {
    // no hay necesidad de un nuevo analisis, terminar programa
    System.out.println("Valores correctos");
    break;
}
else {
    // un nuevo análisis es necesario
    System.out.println("Valores de hierro inadecuados. Diferencia mayor a 10. " +
        "Generando nuevo análisis de seguimmiento");

    analisis = analisis.generarNuevoAnalisis(
        analisis,
        Date.from(Instant.now().plus(7, ChronoUnit.DAYS)));
}

```

(El valor de analisisPendiente es actualizado por la clase GeneraAnalisis al recibir el evento generado por la clase Paciente cuando se le da un nuevo valor a ultimoHierro)

Ejecución



```

Hospital x
/home/kike/.jdk/corretto-17.0.8.1/bin/java -javaagent:/home/kike/bin/idea-I
Análisis 1 del paciente Joe Pris (08-12-2023)
Valores actuales
Hierro: 10
Úrea: 44444
Introducir valor de hierro: 21
Introducir valor de úrea': 3000
*****
Valores de hierro inadecuados.
Diferencia mayor a 10. Generando nuevo análisis de seguimiento
*****
Análisis 2 del paciente Joe Pris (15-12-2023)
Valores actuales
Hierro: 21
Úrea: 3000
Introducir valor de hierro: 20
Introducir valor de úrea': 2000
*****
Valores correctos
*****
Process finished with exit code 0

```

Anexo

Clase Paciente (JavaBean)

```
public class Paciente implements Serializable {

    static final long serialVersionUID = 1L;

    private String nombre;
    private String apellido;
    private String telefono;
    private int edad;
    private int ultimoHierro;
    private int ultimaUrea;

    private final PropertyChangeSupport changeSupport;

    public Paciente() { changeSupport = new PropertyChangeSupport(this); }

    public Paciente(
        String nombre, String apellido, int edad, String telefono, int ultimoHierro, int ultimaUrea)
    {
        this();
        this.nombre = nombre;
        this.apellido = apellido;
        this.telefono = telefono;
        this.edad = edad;
        this.ultimoHierro = ultimoHierro;
        this.ultimaUrea = ultimaUrea;
    }

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        this.changeSupport.addPropertyChangeListener(listener);
    }

    // Setters
    public void setUltimoHierro(int valor) {
        int valorAntiguo = this.ultimoHierro;
        this.ultimoHierro = valor;

        // genera un evento de cambio en el valor de ultimoHierro
        changeSupport.firePropertyChange("ultimoHierro", valorAntiguo, valor);
    }

    public void setNombre(String nombre) { this.nombre = nombre; }

    public void setApellido(String apellido) { this.apellido = apellido; }

    public void setTelefono(String telefono) { this.telefono = telefono; }

    public void setEdad(int edad) { this.edad = edad; }

    public void setUltimaUrea(int valor) { this.ultimaUrea = valor; }

    // Getters
    public String getNombre() { return nombre; }

    public String getApellido() { return apellido; }

    public String getTelefono() { return telefono; }

    public int getEdad() { return edad; }

    public int getUltimoHierro() { return ultimoHierro; }

    public int getUltimaUrea() { return ultimaUrea; }
}
```


Clase GeneraAnálisis (JavaBean)

```

public class GeneraAnálisis implements Serializable, PropertyChangeListener {
    static final long serialVersionUID = 1L;

    private int numAnálisis;
    private Paciente paciente;
    private Date fecha;
    private int análisisPendiente;

    // constructor para cumplir con especificaciones JavaBean
    public GeneraAnálisis() {}

    public GeneraAnálisis(int numAnálisis, Date fecha) {
        this.numAnálisis = numAnálisis;
        this.fecha = fecha;
    }

    public GeneraAnálisis generarNuevoAnálisis(GeneraAnálisis antiguo, Date fecha) {
        GeneraAnálisis nuevoAnálisis = new GeneraAnálisis(
            antiguo.getNumAnálisis() + 1,
            fecha);

        nuevoAnálisis.setPaciente(antiguo.getPaciente());
        return nuevoAnálisis;
    }

    // Getters
    public Date getFecha() { return fecha; }
    public int getNumAnálisis() { return numAnálisis; }
    public Paciente getPaciente() { return paciente; }
    public int getAnálisisPendiente() { return análisisPendiente; }

    //Setters
    public void setPaciente(Paciente paciente) {
        this.paciente = paciente;
        paciente.addPropertyChangeListener(this);
    }

    public void setFecha(Date fecha) { this.fecha = fecha; }
    public void setNumAnálisis(int numAnálisis) { this.numAnálisis = numAnálisis; }
    public void setAnálisisPendiente(int análisisPendiente) { this.análisisPendiente = análisisPendiente; }

    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        if (evt.getPropertyName().equalsIgnoreCase("ultimoHierro")) {
            int variacionHierro = diferenciaEntreValores(evt.getOldValue(), evt.getNewValue());

            if (variacionHierro > 10)
                análisisPendiente++;
        }
    }

    private int diferenciaEntreValores(Object valorAntiguo, Object nuevoValor) {
        Integer antiguo = (Integer) valorAntiguo;
        Integer nuevo = (Integer) nuevoValor;
        return Math.abs(antiguo - nuevo);
    }
}

```

Clase Hospital (Main)

```
public class Hospital {

    private static final DateFormat formatoFecha = new SimpleDateFormat("dd-MM-yyyy");

    public static void main(String[] args) {
        InputUsuario input = new InputUsuario(new Scanner(System.in));

        Paciente paciente = new Paciente("Joe", "Pris", 30, "630009", 10, 44444);

        GeneraAnalisis analisis = new GeneraAnalisis(1, Date.from(Instant.now()));
        analisis.setPaciente(paciente);

        while (true) {
            System.out.println(
                "Análisis " + analisis.getNumAnalisis() +
                " del paciente " + paciente.getNombre() + " " + paciente.getApellido() + " (" +
                formatoFecha.format(analisis.getFecha()) + ")");

            System.out.println("Valores actuales");
            System.out.println("Hierro: " + paciente.getUltimoHierro());
            System.out.println("Úrea: " + paciente.getUltimaUrea());

            int valorHierro = input.solicitarEntero("Introducir valor de hierro: ");
            paciente.setUltimaUrea(valorUrea);

            if (analisis.getAnalisisPendiente() == 0) {
                // no hay necesidad de un nuevo análisis, terminar programa
                System.out.println("*****");
                System.out.println("Valores correctos");
                System.out.println("*****");
                break;
            }
            else {
                // un nuevo análisis es necesario
                System.out.println("*****");
                System.out.println("Valores de hierro inadecuados.");
                System.out.println("Diferencia mayor a 10. Generando nuevo análisis de seguimiento");
                System.out.println("*****");

                analisis = analisis.generarNuevoAnalisis(
                    analisis,
                    Date.from(Instant.now().plus(7, ChronoUnit.DAYS)));
            }
        }
    }
}
```

Clases Util

Clase InputUsuario

La clase input usuario es utilizada para solicitar input al usuario. En este programa, es utilizado para solicitar números enteros

```
/**
 * Solicita una entrada de entero al usuario
 * @param mensaje El mensaje que se imprime al usuario al pedir la entrada
 * @return El entero proporcionado por el usuario
 */
public int solicitarEntero(String mensaje) { return solicitarEntero(mensaje, ValidadorNumeros.sinValidacion()); }

/**
 * Solicita una entrada de entero al usuario, hasta que ésta sea válida
 * @param mensaje El mensaje que se imprime al usuario al pedir la entrada
 * @param validador el objeto que dará validez a la entrada del usuario
 * @return El entero proporcionado por el usuario, una vez validado
 */
public int solicitarEntero(String mensaje, ValidadorNumeros validador) {
    while (true) {
        try {
            // imprime el mensaje al usuario en pantalla y lee el siguiente entero introducido
            System.out.print(mensaje);
            int entero = scanner.nextInt();

            // consume el resto de la línea
            scanner.nextLine();

            // comprueba validez
            if (!validador.validarNumero(entero)) {
                System.out.println(validador.mensajeError());
                continue;
            }

            return entero;
        } catch (InputMismatchException e) {
            resolverInputNumericoInvalido("Por favor, introducir un número entero.");
        }
    }
}
```

Interfaz ValidadorNumeros

Interfaz que expone métodos para validar números según la necesidad del código cliente. También expone clases estáticas predeterminadas que implementan esta interfaz, para facilitar su uso por parte del código cliente.

```
public interface ValidadorNumeros {

    boolean validarNumero(double numero);
    String mensajeError();

    static ValidadorNumeros soloPositivos() { return new ValidadorPositivos(); }
    static ValidadorNumeros sinValidacion() { return new NoValidador(); }
    static ValidadorNumeros enIntervalo(int start, int end) { return new ValidadorEnIntervalo(start, end); }

    /**
     * Clase que valida números solo si son positivos
     */
    class ValidadorPositivos implements ValidadorNumeros {
        @Override
        public boolean validarNumero(double numero) { return numero >= 0; }

        @Override
        public String mensajeError() { return "Por favor, introducir un número positivo"; }
    }

    /**
     * Clase que valida números solo si se encuentran dentro del intervalo establecido
     */
    class ValidadorEnIntervalo implements ValidadorNumeros {

        private final int start;
        private final int end;

        public ValidadorEnIntervalo(int start, int end) {
            this.start = start;
            this.end = end;
        }

        @Override
        public boolean validarNumero(double numero) { return numero >= start && numero <= end; }

        @Override
        public String mensajeError() { return "Por favor, introducir un número entre " + start + " y " + end; }
    }

    /**
     * Clase que ignora la validación de números
     */
    class NoValidador implements ValidadorNumeros {
        @Override
        public boolean validarNumero(double numero) { return true; }

        @Override
        public String mensajeError() { return ""; }
    }
}
```