Innovación en Formación Profesional

Módulo Profesional 06:

Acceso a datos

Actividad UF3

CICLO FORMATIVO DE GRADO SUPERIOR EN

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

MODALIDAD ONLINE

Enrique Verea

Persistencia en BD nativas XML (con DOM y SAX)

Objetivos

Leer y escribir en BD nativas XML con los parsers DOM y SAX

Competencias asociadas:

- Conocer los parsers DOM y SAX, sus ventajas y diferencias.

Metodología	Entrega
Preparación individual	21 noviembre 2023, en PDF + ZIP

Dedicación estimada	Documentos de referencia
8 horas	

Resultados de aprendizaje

- Leer y escribir en BD nativas usando DOM
- Leer y escribir en BD nativas usando SAX

Criterios de evaluación

- Programar soluciones usando los dos parsers (DOM y SAX)
- 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 por qué (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
  - el pdf incluirá este documento con toda la resolución de los ejercicios (con lo comentado anteriormente)
  - 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 😊

PARTE 1 – TEORIA DOM vs SAX

1. ¿Qué es DOM?

Es una librería que facilita la lectura y escritura de archivos XML. Tiene un mejor desempeño con archivos pequeños y medianos, ya que carga todo el contenido del archivo en memoria. El contenido es representado en una estructura de árbol.

2. ¿Qué es SAX?

Es una librería que facilita la lectura de archivos XML. La lectura del archivo es progresiva y es dirigida por eventos, en lugar de leer el archivo entero de una pasada. Está pensada para archivos de gran tamaño que pueden dar problemas de rendimiento con otras librerías como DOM, que cargan el contenido entero del archivo en memoria.

3. ¿Qué librerías son necesarias para trabajar con DOM?

Las librerías **org.w3c.dom** y **javax.xml.parsers**. También son necesarias las librerías **java.io** para abrir streams a los ficheros y la librería **org.xml.sax**, si se quiere atrapar las excepciones de tipo SAXException.

4. ¿Qué librerías son necesarias para SAX?

Las librerías **org.xml.sax** y **javax.xml.parsers**. También la librería **java.io** si se quiere atrapar excepciones de tipo IOException.

5. Busca el significado de las siguientes interfaces de DOM:

- Document: es una representación del documento XML (o HTML) en su totalidad. Conceptualmente, es la raíz de la estructura de árbol que es la representación lógica del documento, por lo que es el punto de entrada a los datos del mismo.

- Element: es un tipo de Nodo que representa un elemento XML (o HTML). Cada Elemento puede tener Atributos asociados a ellos, además de nodos hijos.

- Node: es el principal tipo de dato del modelo DOM y representa un Nodo en la estructura de árbol del Documento. Puede ser de varios tipos, incluyendo: Texto, Elemento, Documento, entre otros.

- NodeList: es una colección ordenada de los nodos de un documento

- Attr: representa un Atributo de un Elemento. Desde el punto de vista del DOM, los Atributos no son Nodos, sino propiedades que describen un Elemento.
- CharacterData: es un tipo de Nodo que representa datos textuales
- DocumentType: es una interfaz que da acceso a la lista de Elementos que forman parte del Documento

PARTE 2 – Persistencia XML con java (DOM)

Ejercicio6.

Este ejercicio utiliza la clase CreadorXML_DOM, que expone el método *escribirContenidoXML* para la escritura de datos de tipo Propiedad en un documento con formato XML. La interfaz Propiedad es una abstracción de datos que puede contener un valor textual u otras Propiedades hijas, además atributos, de tipo Atributo. La clase Atributo es una abstracción de atributos que describen a la Propiedad a la que pertenecen, y están formados por un nombre y un valor textual.

La clase CreadorXML_DOM tiene un único campo, de tipo Document, creado por DOMImplementation

```
public CreadorXML_DOM(String raiz) throws ParserConfigurationException {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    DOMImplementation implementation = builder.getDOMImplementation();

    documento = implementation.createDocument(null, raiz, null);
    documento.setXmlVersion("1.0");
}
```

Para la modificación del DOM, la clase CreadorXML_DOM añade las Propiedades comenzando en el Elemento raíz. Para añadir cada Propiedad primero se crea un Elemento con el nombre de la misma y se le añaden todos los atributos que lo describen. Luego se determina si la Propiedad tiene hijos o no. Si no tiene hijos se entiende que es la hoja final de la rama a la que pertenece y se agrega un nodo de texto con el valor textual de la propiedad al Elemento. Si la Propiedad tiene hijos se repite el procedimiento anterior por cada hijo, de manera recursiva.

```
private void anadirPropiedad(Element elemento, Propiedad propiedad) {
    Element hijo = documento.createElement(propiedad.getNombre());
    anadirAtributos(hijo, propiedad.getAtributos());

    if (propiedad.getPropiedades().isEmpty()) {
        // la propiedad no tiene propiedades hijas, es la última hoja de esta rama y tiene valor textual
        Text texto = documento.createTextNode(propiedad.getValor());
        hijo.appendChild(texto);
    } else
        anadirPropiedades(hijo, propiedad.getPropiedades());

    elemento.appendChild(hijo);
}

private void anadirAtributos(Element nodo, List<Atributo> atributos) {
    for (Atributo atributo : atributos)
        nodo.setAttribute(atributo.getNombre(), atributo.getValor());
}
```

Finalmente, los datos son escritos al fichero mediante la clase Transformer

```
    Source source = new DOMSource(documento);
    Result result = new StreamResult(new File(ruta)); // abre el fichero de destino

    Transformer transformer = TransformerFactory.newInstance().newTransformer();
    transformer.transform(source, result); //escribe el contenido al fichero
}
```

Resultados

Se crean 3 instancias de la clase Persona, que implementa la interfaz Propiedad y se añaden al DOM mediante la invocación del método *escribirContenidoXML*

```
    try {
        CreadorXML_DOM adaptadorXML = new CreadorXML_DOM("personas");
        adaptadorXML.escribirContenidoXML("personas.xml", persona1, persona2, persona3);
    }
    catch (ParserConfigurationException e) {
        System.err.println("No se ha podido crear el gestor de archivos XML. Causa: " + e.getMessage());
        e.printStackTrace();
    }
    catch (TransformerException e) {
        System.err.println("No se ha podido escrbir el contenido en el fichero. Causa: " + e.getMessage());
        e.printStackTrace();
    }
```

personas.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<personas>
    <persona extranjero="NO" mayor_edad="SI">
        <nombre>Cindy</nombre>
        <apellidos>
            <apellido>Nero</apellido>
        </apellidos>
        <nif>74089868Z</nif>
        <fecha_nacimiento>26/02/1992</fecha_nacimiento>
        <nacionalidad>España</nacionalidad>
        <direccion>
            <direccion>Av. Zumalakarregi 38, 1-4</direccion>
            <ciudad>Alpera</ciudad>
            <provincia>Albacete</provincia>
            <codigo_postal>02690</codigo_postal>
        </direccion>
        <info_contacto>
            <telefono>555 555 555</telefono>
            <email>cindynero@gmail.com</email>
        </info_contacto>
    </persona>
    <persona extranjero="SI" mayor_edad="SI">
        <nombre>Elmer</nombre>
```

(...) *(el archivo entero está adjuntado a este documento)*

Ejercicio7.

Este ejercicio utiliza la clase LectorXML_DOM, que tiene dos campos, el nombre del Elemento raíz del documento y un campo de tipo Document, creado por DocumentBuilder

```
public LectorXML_DOM(String raiz, String ruta) throws ParserConfigurationException, IOException, SAXException{
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();

    this.raiz = raiz;
    this.documento = builder.parse(new File(ruta)); //abre fichero para lectura
}
```

La clase expone el método *leerContenido,* que imprime el contenido del documento por salida estándar. Para la impresión del contenido, se imprime el nombre de cada nodo, seguido de sus atributos y finalmente su valor textual, si lo tuviera. Si en vez de contener texto, el nodo tiene otros nodos hijos, se repite el procedimiento de manera recursiva. El texto impreso es indentado tomando en cuenta un valor de tabulación base (2) y la profundidad o distancia del nodo con respecto al nodo raíz. La impresión se inicia con el nodo raíz y una profundidad de 0.

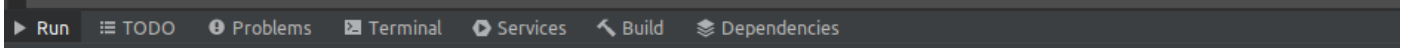
```
public void leerContenido() {
    NodeList personas = documento.getElementsByTagName(raiz);
    Node nodoRaiz = personas.item(0);

    imprimirNodo(nodoRaiz, 0);
}

public void imprimirNodo(Node nodo, int profundidad) {
    if (nodo.getNodeType() != Node.ELEMENT_NODE) // omite nodos que no sean elementos (metadata)
        return;

    int indentacion = TABULACION * profundidad;

    imprimirNombre(nodo, indentacion);
    imprimirAtributos(nodo.getAttributes());

    NodeList hijos = nodo.getChildNodes();
    boolean finalDeRama = hijos.getLength() == 1; // un solo hijo (texto), asume que es el último nodo de la rama

    if (finalDeRama)
        imprimirTexto(nodo);
    else
        imprimirHijos(hijos, profundidad);
}
```

Ejemplo de impresión

```
private void imprimirNombre(Node nodo, int indentacion) {
    System.out.print(" ".repeat(indentacion) + nodo.getNodeName());
}
```

Resultado

```
personas
  persona (extranjero: NO, mayor_edad: SI)
    nombre: Cindy
    apellidos
      apellido: Nero
    nif: 74089868Z
    fecha_nacimiento: 26/02/1992
    nacionalidad: España
    direccion
      direccion: Av. Zumalakarregi 38, 1-4
      ciudad: Alpera
      provincia: Albacete
      codigo_postal: 02690
    info_contacto
      telefono: 555 555 555
      email: cindynero@gmail.com
  persona (extranjero: SI, mayor_edad: SI)
    nombre: Elmer
    apellidos
      apellido: Curio
    nif: 24686643F
    fecha_nacimiento: 08/10/1984
    nacionalidad: Egipto
    direccion
      direccion: Ctra. de Siles 56, 2-3
      ciudad: Quiroga
      provincia: Lugo
      codigo_postal: 27320
    info_contacto
      telefono: 888 888 888
      email: elmercurio@gmail.com
  persona (extranjero: NO, mayor_edad: SI)
    nombre: Esteban
    apellidos
      apellido: Dido
      apellido: Arrimado
    nif: 827196426
    fecha_nacimiento: 19/04/1998
```



(...)

```
      direccion: C/ Amoladera 80, 4-6
      ciudad: Somosierra
      provincia: Madrid
      codigo_postal: 28756
    info_contacto
      telefono: 222 222 222
      email: elbandido@gmail.com

Process finished with exit code 0
```

PARTE 3 – Acceso a XML con SAX

Ejercicio8.

Este ejercicio utiliza la clase SAXHandler, que extiende DefaultHandler y responde a los siguientes eventos:

startDocument:	Imprime ‘Comienzo del Documento XML’
startElement:	Imprimer ‘Principio Elemento’ y el nombre del Elemento
characters:	Imprime ‘Caracteres’ y el valor de los caracteres
endElement:	Imprime ‘Fin Elemento’ y el nombre del elemento
endDocument:	Imprime ‘Final del Documento XML’

```java
private static final int INDENT = 8;
private static final int INDENT_FIN_ELEMENTO = 7;

@Override
public void startDocument() throws SAXException {
    super.startDocument();
    System.out.println("Comienzo del documento XML");
}

private void imprimirIndentado(String contenido, int indent) { System.out.println(" ".repeat(indent) + contenido); }

@Override
public void startElement(String uri, String localName, String name, Attributes attributes) {
    imprimirIndentado("Principio Elemento: " + name, INDENT);
}

@Override
public void characters(char[] ch, int start, int length) {
    String caracteres = new String(ch, start, length).strip();
    imprimirIndentado("Caracteres: " + caracteres, INDENT);
}

@Override
public void endElement(String uri, String localName, String qName) {
    imprimirIndentado("Fin Elemento: " + qName, INDENT_FIN_ELEMENTO);
}

@Override
public void endDocument() throws SAXException {
    super.endDocument();
    System.out.println("Final del Documento XML");
}
```

Resultado

```
Comienzo del documento XML
        Principio Elemento: galaxia
        Caracteres:
        Principio Elemento: estrellas
        Caracteres:
        Principio Elemento: estrella
        Caracteres:
        Principio Elemento: planeta
        Caracteres: Tierra
       Fin Elemento: planeta
        Caracteres:
        Principio Elemento: satelite
        Caracteres: Luna
       Fin Elemento: satelite
        Caracteres:
        Principio Elemento: cometa
        Caracteres: Halley
       Fin Elemento: cometa
        Caracteres:
       Fin Elemento: estrella
        Caracteres:
        Principio Elemento: estrella
        Caracteres:
        Principio Elemento: planeta
        Caracteres: VegaPlanet
       Fin Elemento: planeta
```

(...)

```
       Fin Elemento: satelite
        Caracteres:
        Principio Elemento: cometa
        Caracteres: SantaExpress
       Fin Elemento: cometa
        Caracteres:
       Fin Elemento: estrella
        Caracteres:
       Fin Elemento: estrellas
        Caracteres:
       Fin Elemento: galaxia
Final del Documento XML

Process finished with exit code 0
```

Ejercicio9.

Este ejercicio utiliza la clase PersonaSAXHandler, que extiende la clase DefaultHandler y mantiene como estado la línea (elemento) que se encuentra en construcción y la profundidad con respecto al elemento raíz del documento. PesonaSAXHandler responde a los siguientes eventos:

startElement:	Realiza las siguientes acciones: ● Aumenta el valor de la profundidad en 1 ● Si la profundidad > 0, una línea se encuentra en construcción, así que la consume ● Crea una nueva línea con el nombre y atributos del elemento
characters:	Añade los caracteres a la línea en construcción
endElement:	Disminuye el valor de la profundidad en 1
endDocument:	Consume la última línea en construcción

Consumir línea

Para PersonaSAXHandler, consumir una línea significa imprimir el texto de la misma por salida estándar y dar al estado *línea* un nuevo valor de cadena de texto vacía.

```
private void construirLinea(String contenido) {
    int indentado = TABULACION * profundidad;
    linea = " ".repeat(indentado) + contenido;
}
```

Eventos

```
@Override
public void startElement(String uri, String localName, String name, Attributes attributes) {
    profundidad++;

    if (profundidad > 0)
        consumirLinea();

    String elemento = name+": ";

    if (attributes.getLength() > 0)
        elemento += formatearAtributos(attributes);

    construirLinea(elemento);
}
```

```java
@Override
public void characters(char[] ch, int start, int length) {
    String caracteres = new String(ch, start, length).strip();

    if (!caracteres.isBlank())
        anadirALinea(caracteres);
}

@Override
public void endElement(String uri, String localName, String qName) {
    profundidad--;
}

@Override
public void endDocument() throws SAXException {
    super.endDocument();
    consumirLinea();
}
```

Resultado

```java
public class Ejercicio9 {

    public static void main(String[] args) {
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser parser = factory.newSAXParser();
            PersonasSAXHandler handler = new PersonasSAXHandler();

            parser.parse("personas.xml", handler);

        } catch (ParserConfigurationException | SAXException | IOException e) {
            System.err.println("Error durante la lectura SAX. Causa: " + e.getMessage());
        }
    }
}
```

```
personas:
  persona: (extranjero: NO, mayor_edad: SI)
    nombre: Cindy
    apellidos:
      apellido: Nero
    nif: 74089868Z
    fecha_nacimiento: 26/02/1992
    nacionalidad: España
    direccion:
      direccion: Av. Zumalakarregi 38, 1-4
      ciudad: Alpera
      provincia: Albacete
      codigo_postal: 02690
    info_contacto:
      telefono: 555 555 555
      email: cindynero@gmail.com
  persona: (extranjero: SI, mayor_edad: SI)
    nombre: Elmer
    apellidos:
      apellido: Curio
    nif: 24686643F
    fecha_nacimiento: 08/10/1984
    nacionalidad: Egipto
    direccion:
```

(...)

```
    apellidos:
      apellido: Dido
      apellido: Arrimado
    nif: 827196426
    fecha_nacimiento: 19/04/1998
    nacionalidad: España
    direccion:
      direccion: C/ Amoladera 80, 4-6
      ciudad: Somosierra
      provincia: Madrid
      codigo_postal: 28756
    info_contacto:
      telefono: 222 222 222
      email: elbandido@gmail.com

Process finished with exit code 0
```

Anexo

Ejercicio 6

datos package

Interfaz Propiedad

Abstracción de datos en forma de Propiedad. Expone métodos *getters* para el nombre, atributos, valor y propiedades hijas.

```java
public interface Propiedad {

    String getNombre();
    String getValor();
    List<Propiedad> getPropiedades();
    List<Atributo> getAtributos();
```

Clase PropiedadSimple

Implementa la interfaz Propiedad. Representa una Propiedad sin propiedades hijas, con valor textual.

```java
class PropiedadSimple implements Propiedad {
    private final String nombre;
    private final String valor;
    private final List<Atributo> atributos = new ArrayList<>();

    public PropiedadSimple(String nombre, String valor) {
        this.nombre = nombre;
        this.valor = valor;
    }

    /**
     * Da acceso al nombre de esta instancia de Propiedad
     * @return el nombre de la Propiedad
     */
    @Override
    public String getNombre() { return nombre; }

    /**
     * Da acceso al valor de esta instancia de Propiedad
     * @return el valor de la Propiedad
     */
    @Override
    public String getValor() { return valor; }

    /**
     * Da acceso a las propiedades hijas de esta instancia de Propiedad.
     * Al ser una Propiedad simple, regresa una lista vacía
     * @return Una lista vacía
     */
    @Override
    public List<Propiedad> getPropiedades() { return List.of(); }

    /**
     * Da acceso a los atributos que describen esta instancia de Propiedad
     * @return Una lista con todos los atributos de esta Propiedad
     */
    @Override
    public List<Atributo> getAtributos() { return atributos; }
}
```

Clase abstracta PropiedadCompuesta

Implementa la interfaz Propiedad. Representa una Propiedad con propiedades hijas, sin valor textual

```java
public abstract class PropiedadCompuesta implements Propiedad {

    private final String nombre;
    private final List<Atributo> atributos = new ArrayList<>();

    public PropiedadCompuesta(String nombre) { this.nombre = nombre; }

    /**
     * Da acceso al nombre de esta instancia de Propiedad
     * @return el nombre de la Propiedad
     */
    @Override
    public String getNombre() { return nombre; }

    protected void anadirAtributo(Atributo atributo) { atributos.add(atributo); }

    /**
     * Da acceso al valor de esta instancia de Propiedad
     * @return el valor de la Propiedad
     */
    @Override
    public String getValor() {
        throw new UnsupportedOperationException("Propiedades con hijos no tienen un valor textual");
    }

    /**
     * Da acceso a los atributos que describen esta instancia de Propiedad
     * @return Una lista con todos los atributos de esta Propiedad
     */
    @Override
    public List<Atributo> getAtributos() { return atributos; }
}
```

Clase Atributo

Abstracción de atributos que describen a la clase Propiedad. Tiene un nombre y un valor textual.

```java
public class Atributo {

    private final String nombre;
    private final String valor;

    public Atributo(String nombre, String valor) {
        this.nombre = nombre;
        this.valor = valor;
    }

    public Atributo(String nombre, boolean valor) { this(nombre, valor ? "SI" : "NO"); }

    public String getNombre() { return nombre; }

    public String getValor() { return valor; }
}
```

persona package

Clase Persona

Extiende la clase PropiedadCompuesta. Está compuesta por las siguientes Propiedades hijas: nombre, apellidos, nif, fecha de nacimiento, dirección, nacionalidad, información de contacto. Se construye con una interfaz fluida (Builder)

```
public class Persona extends PropiedadCompuesta {

    private final Propiedad nombre;
    private final Propiedad apellidos;
    private final Propiedad fechaNacimiento;
    private final Propiedad nif;
    private final Propiedad direccion;
    private final Propiedad nacionalidad;
    private final Propiedad infoDeContacto;

    public Persona(Builder builder) {
        super("persona");

        validarConstruccion(builder);

        String fechaDeNacimiento = DateTimeFormatter.ofPattern("dd/MM/yyyy").format(builder.fechaNacimiento);
        long anosEntreFechaDeNacimientoYHoy = ChronoUnit.YEARS.between(builder.fechaNacimiento, LocalDate.now());

        boolean mayorDeEdad = anosEntreFechaDeNacimientoYHoy >= 18;
        boolean extranjero = !builder.nacionalidad.equalsIgnoreCase("España");

        anadirAtributo(new Atributo("mayor_edad", mayorDeEdad));
        anadirAtributo(new Atributo("extranjero", extranjero));

        nombre = Propiedad.simple("nombre", builder.nombre);
        nif = Propiedad.simple("nif", builder.nif);
        nacionalidad = Propiedad.simple("nacionalidad", builder.nacionalidad);
        fechaNacimiento = Propiedad.simple("fecha_nacimiento", fechaDeNacimiento);

        apellidos = new Apellidos(builder);
        direccion = new Direccion(builder);
        infoDeContacto = new InfoDeContacto(builder);
    }
```

Clase Apellidos

Extiende la clase PropiedadCompuesta. Es Propiedad hija de la clase Persona. Está compuesta por una cantidad arbitraria de Propiedades *simples*, que representan cada apellido de la Persona.

```java
public class Apellidos extends PropiedadCompuesta {

    private final List<Propiedad> apellidos;

    Apellidos(Persona.Builder builder) {
        super("apellidos");
        apellidos = Arrays
                .stream(builder.apellidos)
                .map(apellido -> Propiedad.simple("apellido", apellido))
                .collect(Collectors.toList());
    }

    /**
     * Da acceso a las propiedades hijas de esta instancia de Propiedad
     * @return Una lista con las Propiedades hijas de esta instancia de Propiedad
     */
    @Override
    public List<Propiedad> getPropiedades() { return new ArrayList<>(apellidos); }
}
```

Clase Dirección

Extiende la clase PropiedadCompuesta. Es Propiedad hija de la clase Persona. Está compuesta por las Propiedades *simples* dirección, ciudad, provincia y código postal.

```java
class Direccion extends PropiedadCompuesta {

    private final Propiedad direccion;
    private final Propiedad ciudad;
    private final Propiedad provincia;
    private final Propiedad codigoPostal;

    Direccion(Persona.Builder builder) {
        super("direccion");
        this.direccion = Propiedad.simple("direccion", builder.direccion);
        this.ciudad = Propiedad.simple("ciudad", builder.ciudad);
        this.provincia = Propiedad.simple("provincia", builder.provincia);
        this.codigoPostal = Propiedad.simple("codigo_postal", builder.codigoPostal);
    }

    /**
     * Da acceso a las propiedades hijas de esta instancia de Propiedad
     * @return Una lista con las Propiedades hijas de esta instancia de Propiedad
     */
    @Override
    public List<Propiedad> getPropiedades() { return List.of(direccion, ciudad, provincia, codigoPostal); }

}
```

Clase InfoDeContacto

Extiende la clase PropiedadCompuesta. Es Propiedad hija de la clase Persona. Está compuesta por las Propiedades *simples* email y teléfono.

```java
public class InfoDeContacto extends PropiedadCompuesta {

    private final Propiedad telefono;
    private final Propiedad email;

    InfoDeContacto(Persona.Builder builder) {
        super("info_contacto");
        telefono = Propiedad.simple("telefono", builder.telefono);
        email = Propiedad.simple("email", builder.email);
    }

    /**
     * Da acceso a las propiedades hijas de esta instancia de Propiedad
     * @return Una lista con las Propiedades hijas de esta instancia de Propiedad
     */
    @Override
    public List<Propiedad> getPropiedades() { return List.of(telefono, email); }
}
```

Clase CreadorXML_DOM

Clase que crea documentos XML a través de la interfaz DOMImplementation y escribe el contenido de clases que implementen la interfaz Propiedad en el mismo.

```java
public class CreadorXML_DOM {

    private final Document documento;

    public CreadorXML_DOM(String raiz) throws ParserConfigurationException {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        DOMImplementation implementation = builder.getDOMImplementation();

        documento = implementation.createDocument(null, raiz, null);
        documento.setXmlVersion("1.0");
    }

    /**
     * Escribe en un archivo con la ruta dada, el contenido de las propiedades pasadas a este método, con fo
     * @param ruta la ruta del archivo
     * @param datos representación de los datos, estructurados en forma de árbol
     * @throws TransformerException condición excepcional durante el proceso escritura de datos
     */
    public void escribirContenidoXML(String ruta, Propiedad... datos) throws TransformerException {
        Element raiz = documento.getDocumentElement();
        anadirPropiedades(raiz, Arrays.asList(datos));

        Source source = new DOMSource(documento);
        Result result = new StreamResult(new File(ruta)); // abre el fichero de destino

        Transformer transformer = TransformerFactory.newInstance().newTransformer();
        transformer.transform(source, result); //escribe el contenido al fichero
    }

    private void anadirPropiedades(Element elemento, List<Propiedad> propiedades) {
        for (Propiedad propiedad : propiedades)
            anadirPropiedad(elemento, propiedad);
    }

    private void anadirPropiedad(Element elemento, Propiedad propiedad) {
        Element hijo = documento.createElement(propiedad.getNombre());
        anadirAtributos(hijo, propiedad.getAtributos());

        if (propiedad.getPropiedades().isEmpty()) {
            // la propiedad no tiene propiedades hijas, es la última hoja de esta rama y tiene valor textual
            Text texto = documento.createTextNode(propiedad.getValor());
            hijo.appendChild(texto);
        } else
            anadirPropiedades(hijo, propiedad.getPropiedades());

        elemento.appendChild(hijo);
    }

    private void anadirAtributos(Element nodo, List<Atributo> atributos) {
        for (Atributo atributo : atributos)
            nodo.setAttribute(atributo.getNombre(), atributo.getValor());
    }
}
```

Clase Main

```
public class Ejercicio6 {

    public static void main(String[] args) {
        Persona persona1 =
                Persona.conNombre("Cindy")
                        .appellidos("Nero")
                        .nif("74089868Z")
                        .nacionalidad("España")
                        .fechaNacimiento(LocalDate.of(1992, Month.FEBRUARY, 26))
                        .email("cindynero@gmail.com")
                        .telefono("555 555 555")
                        .direccion("Av. Zumalakarregi 38, 1-4")
                        .provincia("Albacete")
                        .ciudad("Alpera")
                        .codigoPostal("02690")
                        .construir();

        Persona persona2 =
                Persona.conNombre("Elmer")
                        .appellidos("Curio")
                        .nif("24686643F")
                        .nacionalidad("Egipto")
                        .fechaNacimiento(LocalDate.of(1984, Month.OCTOBER, 8))
                        .email("elmercurio@gmail.com")
                        .telefono("888 888 888")
                        .direccion("Ctra. de Siles 56, 2-3")
                        .provincia("Lugo")
                        .ciudad("Quiroga")
                        .codigoPostal("27320")
                        .construir();

        Persona persona3 =
                Persona.conNombre("Esteban")
                        .appellidos("Dido", "Arrimado")
                        .nif("827196426")
                        .nacionalidad("España")
                        .fechaNacimiento(LocalDate.of(1998, Month.APRIL, 19))
                        .email("elbandido@gmail.com")
                        .telefono("222 222 222")
                        .direccion("C/ Amoladera 80, 4-6")
                        .provincia("Madrid")
                        .ciudad("Somosierra")
                        .codigoPostal("28756")
                        .construir();

        try {
            CreadorXML_DOM adaptadorXML = new CreadorXML_DOM("personas");
            adaptadorXML.escribirContenidoXML("personas.xml", persona1, persona2, persona3);
        }
        catch (ParserConfigurationException e) {
            System.err.println("No se ha podido crear el gestor de archivos XML. Causa: " + e.getMessage());
            e.printStackTrace();
        }
        catch (TransformerException e) {
            System.err.println("No se ha podido escrbir el contenido en el fichero. Causa: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Ejercicio 7

Clase LectorXML_DOM

Clase que carga el contenido de archivos XML a través de la interfaz DocumentBuilder y lo imprime por salida estándar.

```java
public class LectorXML_DOM {

    private static final int TABULACION = 2;

    private final Document documento;
    private final String raiz;

    public LectorXML_DOM(String raiz, String ruta) throws ParserConfigurationException, IOException, SAXException{
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();

        this.raiz = raiz;
        this.documento = builder.parse(new File(ruta)); //abre fichero para lectura
    }

    /**
     * Lee el contenido del documento XML de la ruta pasada a esta instancia y lo imprime por salida estándar
     */
    public void leerContenido() {
        NodeList personas = documento.getElementsByTagName(raiz);
        Node nodoRaiz = personas.item(0);

        imprimirNodo(nodoRaiz, 0);
    }
    private void imprimirNodo(Node nodo, int profundidad) {
        if (nodo.getNodeType() != Node.ELEMENT_NODE) // omite nodos que no sean elementos (metadata)
            return;

        int indentacion = TABULACION * profundidad;

        imprimirNombre(nodo, indentacion);
        imprimirAtributos(nodo.getAttributes());

        NodeList hijos = nodo.getChildNodes();
        boolean finalDeRama = hijos.getLength() == 1; // un solo hijo (texto), asume que es el último nodo de la rama

        if (finalDeRama)
            imprimirTexto(nodo);
        else
            imprimirHijos(hijos, profundidad);
    }

    private void imprimirHijos(NodeList hijos, int profundidad) {
        System.out.println();
        for (int i = 0; i < hijos.getLength(); i++)
            imprimirNodo(hijos.item(i), profundidad + 1);
    }

    private void imprimirNombre(Node nodo, int indentacion) {
        System.out.print(" ".repeat(indentacion) + nodo.getNodeName());
    }
```

```
    private void imprimirAtributos(NamedNodeMap atributos) {
        if (atributos == null || atributos.getLength() == 0)
            return;

        System.out.print(" (");
        for (int i = 0; i < atributos.getLength(); i++) {
            imprimirAtributo(atributos.item(i));

            if (i < atributos.getLength() - 1) // añade coma para separar atributos, excepto el último
                System.out.print(", ");
        }
        System.out.print(")");
    }

    private void imprimirAtributo(Node atributo) {
        System.out.print(atributo.getNodeName()+": " + atributo.getTextContent());
    }

    private void imprimirTexto(Node nodo) { System.out.println(": " + nodo.getTextContent()); }
}
```

Clase Main

```
public class Ejercicio7 {

    public static void main(String[] args) {
        try {
            LectorXML_DOM lectorXML = new LectorXML_DOM("personas", "personas.xml");
            lectorXML.leerContenido();
        }
        catch (ParserConfigurationException e) {
            System.err.println("No se ha podido crear el gestor de archivos XML. Causa: " + e.getMessage());
            e.printStackTrace();
        }
        catch (SAXException | IOException e) {
            System.err.println("No se ha podido escrbir el contenido en el fichero. Causa: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Ejercicio 8

Clase SAXHandler

Clase que extiende la clase DefaultHandler y responde a los eventos de la misma durante la lectura de archivos XML.

```java
public class SAXHandler extends DefaultHandler {

    private static final int INDENT = 8;
    private static final int INDENT_FIN_ELEMENTO = 7;

    private void imprimirIndentado(String contenido, int indent) { System.out.println(" ".repeat(indent) + contenido); }

    /**
     * Comienzo del documento
     */
    @Override
    public void startDocument() throws SAXException {
        super.startDocument();
        System.out.println("Comienzo del documento XML");
    }

    /** Comienzo de un elemento
     * @param name el nombre del elemento
     */
    @Override
    public void startElement(String _s, String _l, String name, Attributes _a) {
        imprimirIndentado("Principio Elemento: " + name, INDENT);
    }

    /**
     * Contenido en caracteres de un elemento
     * @param ch Los caracteres
     * @param start La posición inicial del array de caracteres
     * @param length El número de caracteres del array que pertenecen al elemento
     */
    @Override
    public void characters(char[] ch, int start, int length) {
        String caracteres = new String(ch, start, length).strip();
        imprimirIndentado("Caracteres: " + caracteres, INDENT);
    }
    /**
     * Final de un elemento
     * @param name El nombre del elemento
     */
    @Override
    public void endElement(String _u, String _l, String name) {
        imprimirIndentado("Fin Elemento: " + name, INDENT_FIN_ELEMENTO);
    }

    /**
     * Final del documento
     */
    @Override
    public void endDocument() throws SAXException {
        super.endDocument();
        System.out.println("Final del Documento XML");
    }
```

Clase Main

```
public class Ejercicio8 {

    public static void main(String[] args) {
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser parser = factory.newSAXParser();
            SAXHandler handler = new SAXHandler();

            parser.parse("galaxias.xml", handler);

        } catch (ParserConfigurationException | SAXException | IOException e) {
            System.err.println("Error durante la lectura SAX. Causa: " + e.getMessage());
        }
    }
}
```

Ejercicio 9

Clase PersonasSAXHandler

Clase que extiende la clase DefaultHandler y responde a los eventos de la misma durante la lectura de archivos XML que contienen datos tipo Persona.

```
public class PersonasSAXHandler extends DefaultHandler {

    private static final int TABULACION = 2;

    private int profundidad = -1;
    private String linea;

    private void anadirALinea(String contenido) { linea += contenido; }

    private void consumirLinea() {
        System.out.println(linea);
        linea = "";
    }

    /** Comienzo de un elemento
     * @param name el nombre del elemento
     * @param attributes los atributos que describen este elemento
     */
    @Override
    public void startElement(String uri, String localName, String name, Attributes attributes) {
        profundidad++;

        if (profundidad > 0)
            consumirLinea();

        String elemento = name+": ";

        if (attributes.getLength() > 0)
            elemento += formatearAtributos(attributes);

        construirLinea(elemento);
    }
```

```java
    private void construirLinea(String contenido) {
        int indentado = TABULACION * profundidad;
        linea = " ".repeat(indentado) + contenido;
    }

    private String formatearAtributos(Attributes atributos) {

        StringBuilder resultado = new StringBuilder("(");
        for (int i = 0; i < atributos.getLength(); i++) {

            String nombreAtributo = atributos.getQName(i);
            String valorAtributo = atributos.getValue(i);

            resultado.append(nombreAtributo);
            resultado.append(": ");
            resultado.append(valorAtributo);

            if (i < atributos.getLength() -1) // añade una coma para separar los atributos, excepto el último
                resultado.append(", ");

        }
        resultado.append(")");

        return resultado.toString();
    }

    /**
     * Contenido en caracteres de un elemento
     * @param ch Los caracteres
     * @param start La posición inicial del array de caracteres
     * @param length El número de caracteres del array que pertenecen al elemento
     */
    @Override
    public void characters(char[] ch, int start, int length) {
        String caracteres = new String(ch, start, length);

        if (!caracteres.isBlank())
            anadirALinea(caracteres);
    }

    /**
     * Final de un elemento
     */
    @Override
    public void endElement(String uri, String localName, String qName) { profundidad--; }

    /**
     * Final del documento
     */
    @Override
    public void endDocument() throws SAXException {
        super.endDocument();
        consumirLinea();
    }
```

Clase Main

```java
public class Ejercicio9 {

    public static void main(String[] args) {
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser parser = factory.newSAXParser();
            PersonasSAXHandler handler = new PersonasSAXHandler();

            parser.parse("personas.xml", handler);

        } catch (ParserConfigurationException | SAXException | IOException e) {
            System.err.println("Error durante la lectura SAX. Causa: " + e.getMessage());
        }
    }
}
```