INTRODUCCIÓN – MÓDULO ACCESO A DATOS

1. Acceso a datos en una aplicación
2. Ficheros
3. Base de datos
4. Desarrollo de componentes

1 ACCESO A DATOS

¿Cuando hablamos de acceso a datos en una aplicación informática ?

Las aplicaciones informáticas se pueden diferenciar, en dos partes:

1. El programa o algoritmo propiamente dicho, que realiza las operaciones deseadas con los datos necesarios.
2. Los datos con los que opera le programa. Esos datos pueden ser obtenidos por el programa mediante diversos métodos:
   Mediante teclado
   Escaneados
   Soporte de almacenamiento secundario, etc.

   Cuando realizamos un programa, nos interesa que el programa guarde los datos que hemos introducido, o los resultados que dicho programa haya obtenido, de manera que si el programa termina su ejecución, l**os datos no se pierdan y puedan ser recuperados posteriormente, es decir, persistan**. Para ello se utilizan ficheros o bases de datos que se guardan en un dispositivo de memoria NO volátil (un disco duro).

   Los datos que se guardan en un almacenamiento secundario como ficheros, base de datos se denominan **datos persistentes.**

La mayoría de las veces el almacenamiento secundario suele ser una base de datos relacional.

El módulo de acceso a datos consiste en **conocer los tipos de almacenamiento de datos y cómo accedemos a ellos para trabajar con los datos en el programa.**

¿Cuáles son las estrategias de acceso para gestionar la persistencia de datos?

Hay diversas estrategias de acceso a datos para gestionar la persistencia de los datos:

- **Mediante ficheros.**
- **Bases de datos** Trabajar con base de datos tienen muchas ventajas que trabajar con ficheros:

  ➔ Evitar datos repetidos o duplicados
  ➔ Eliminar información inconsistente.
  ➔ Compartir datos globalmente.
  ➔ Centralizar la información. Las BDs permiten tener los datos centralizados en un solo lugar, puede ser un servidor.
  ➔ Garantizar el mantenimiento de la **integridad en la información**. Corrección y exactitud de la información contenida en una base de datos.
  ➔ Acceso rápidos a los datos
  ➔ **Independencia de datos.** La independencia de datos implica una separación entre programas y datos, es decir, se pueden hacer cambios en la información que contiene la base de datos, o tener acceso a la base de datos de diferente manera, sin tener que hacer cambios en las aplicaciones o en los programas.

¿Qué método de acceso a datos es mejor? ¿Cuál deberías utilizar en la próxima aplicación que construyas?

No hay un método que sea el mejor de todos. Tenemos que tener claro que tipo de aplicación hay que construir y según eso, estudiar el tipo de sistema de almacenamiento será mejor usar.

Ejemplo: si voy a crear una base de datos para guardar mi colección de vídeos, probablemente no me va a interesar utilizar una base de datos Oracle, sino un producto mucho más barato, y sencillo de instalar y mantener.

2 FICHEROS

Antes de la aparición de las bases de datos, la información se guardaba en ficheros. Por tanto el programador de las aplicaciones que usaran ese fichero tenía que conocer detalladamente el formato de los datos (posiciones en los datos) para poder extraer la información de forma correcta. Cuando aparecieron las bases de datos se empezó a dejar de usar los ficheros.

Sin embargo en algunas aplicaciones modernas se necesita un simple fichero para guardar información (fichero de configuración, fichero log).

En java existen diversas clases para el manejo de fichero y son muy útiles para guardar poca información.

Actualmente los ficheros se usan para guardar datos siguiendo un patrón o estructura bien definida como por ejemplo en ficheros **XML**.

```
<?xml version="1.0" encoding="UTF-8"?>
<selva>
   <animal>
      <nombre>Werthers</nombre>
      <tipo>Pantera</tipo>
      <color>Negro</color>
      <edad>12</edad>
      </animal>
   <animal>
      <nombre>Bun</nombre>
      <tipo>León</tipo>
      <color>Marrón</color>
      <edad>15</edad>
   </animal>
</selva>
```

Los ficheros **XML** facilitan el intercambio de datos porque los datos XML están almacenados en formato de texto simple, lo que permite almacenar y compartir la información entre diferentes sistemas o aplicaciones de manera más rápida y sencilla. Además los datos XML pueden ser leídos por diferentes plataformas o sistemas que, aún siendo incompatibles, permiten a los desarrolladores intercambiar fácilmente los datos entre una y otra.

Cuando se necesita intercambiar información a través de varias plataformas de hardware o de software, o de varias aplicaciones. A veces se exporta de una base de datos a ficheros XML para trasladar la información a otra base de datos que leerá esos ficheros XML.

3 BASES DE DATOS

Un sistema de información orientado hacia los datos, que pretende recuperar y almacenar la información de manera eficiente y cómoda.
Surge en un intento de resolver las dificultades del procesamiento tradicional de datos, teniendo en cuenta que los datos suelen ser independientes de las aplicaciones.

Las ventajas que aportan los sistemas de bases de datos respecto a los sistemas de archivos convencionales son:

- Independencia de los datos respecto de los procedimientos.
- El usuario tiene una visión abstracta de los datos, sin necesidad de ningún conocimiento sobre la implementación de los ficheros de datos, índices, etc.
- Disminución de las redundancias y en consecuencia,
- Disminución de la posibilidad de que se produzca inconsistencia de datos.
- Mayor disponibilidad de los datos.
- Mayor seguridad de los datos.
- Mayor privacidad de los datos.
- Compartición de los datos. Los datos pueden ser accedidos por varios usuarios simultáneamente teniendo previstos procedimientos para salvaguardar la integridad de los mismos.

Base de datos

3.1 BASE DE DATOS ORIENTADA A OBJETOS

Una base de datos orientada a objetos es aquella que implementa dentro de un sistema de información la representación de datos en forma de objetos. Estas bases de datos se diferencian de las bases de datos relacionales clásicas, debido a que **no responden a un sistema de tablas para registrar su información.**
La mayoría de las bases de objetos ofrecen lenguajes de consultas para encontrar los objetos que pertenecen a la base.

- Los sistemas de bases de datos orientadas a objetos **soportan un modelo de objetos puro,** ya que no están basados en extensiones de otros modelos más clásicos como el relacional.

- El acceso a los datos puede ser más rápido con las bases de datos orientadas a objetos que con las bases de datos tradicionales porque no hay necesidad de utilizar las uniones o joins, que sí se necesitan en los esquemas relacionales

tabulares. Esto se debe a que **un objeto puede recuperarse directamente sin una búsqueda, simplemente siguiendo punteros.** Las bases de datos son capaces de funcionar con varios tipos de lenguajes de programación conocidos.

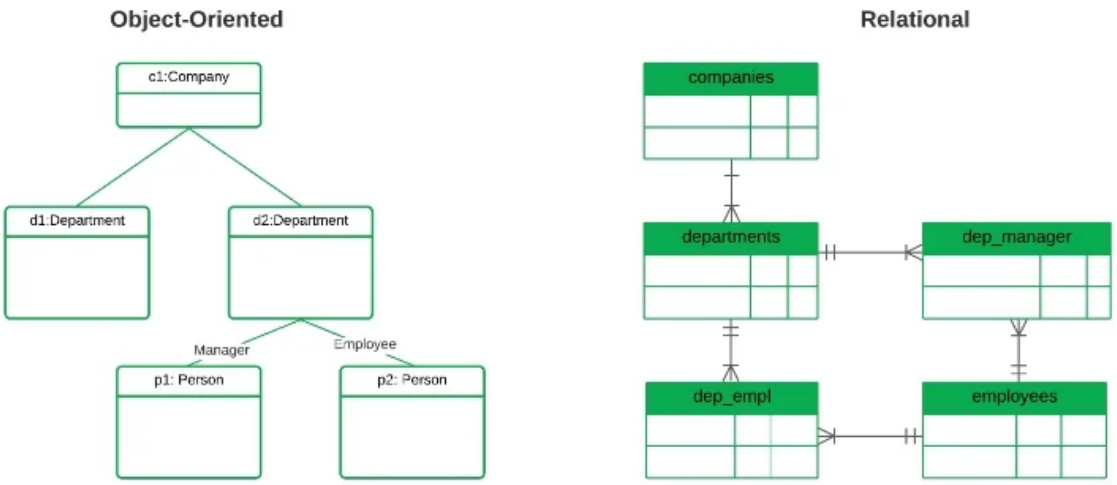
- Entre los aspectos más positivos que poseen dichas bases de datos se encuentra su **lenguaje de consulta.** Estas implementan un sistema declarativo que permiten encontrar y recuperar los objetos de la base de datos. Las bases de datos orientadas a objetos **no utilizan SQ**L e incluyen un lenguaje específico para hacer las consultas.
  Permiten mayor capacidad de modelado. El modelado de datos orientado a objetos permite modelar el mundo real de una manera mucho más fiel. Esto se debe a que un objeto permite encapsular tanto un estado como un comportamiento. También puede almacenar todas las relaciones que tenga con otros objetos. Los objetos pueden agruparse para formar objetos complejos(herencia).

- Extensibilidad: se pueden construir nuevos tipos de datos a partir de los ya existentes.
- Reusabilidad de clases. Menor tiempo de desarrollo.
- Reducir la redundancia.

Desventajas de las bases de datos orientadas a objetos frente a las relacionales.

Como desventajas o puntos débiles de las BBDDOO respecto a las relacionales podemos mencionar:

- La reticencia del mercado, tanto para desarrolladores como usuarios, a este tipo de bases de datos.
- Carencia de experiencia. Tecnología relativamente nueva. No se dispone del nivel de experiencia del que dispone para los sistemas relacionales.
- Panorama actual. Tanto los sistemas gestores de bases de datos como los sistemas gestores de bases de datos objeto-relacionales están muy extendidos. **SQL** es un estándar aprobado y **ODBC y JDBC** son estándares también. Además, el modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales.
- Dificultades en optimización. La optimización de consultas necesita realizar una compresión de la implementación de los objetos, para poder acceder a la base de datos de manera eficiente. Sin embargo, esto compromete el concepto de encapsulación.



3.2 BASE DE DATOS OBJETO-RELACIONALES

La base de datos objeto-relacional es una extensión de la base de datos relacional tradicional, a la cual se le proporcionan características de la programación orientada a objetos (POO)

En una **Base de Datos Objeto Relacional** el diseñador puede crear sus propios tipos de datos y crear métodos para esos tipos de datos.

Por ejemplo, podríamos definir un tipo de la siguiente manera:

```
CREATE TYPE direccion_t AS OBJECT (
calle VARCHAR2(200),
ciudad VARCHAR2(200),
prov CHAR(2),
codpos VARCHAR2(20));

CREATE TYPE cliente_t AS OBJECT (
clinum NUMBER,
clinomb VARCHAR2(200),
direccion direccion_t,
telefono VARCHAR2(20),
fecha_nac DATE,
MEMBER FUNCTION edad R
```

Otro ejemplo:

CREATE TYPE Persona (nombre VARCHAR(20), direccion VARCHAR(20))

Con esto se necesita definir varios tipos de personas:

CREATE TYPE Estudiante UNDER Persona (curso VARCHAR(20), departamento VARCHAR(20));
CREATE TYPE Profesor UNDER Persona (sueldo INTEGER, departamento VARCHAR(20)) ;

El objetivo de este concepto es poder aplicar la tecnología madura de bases de datos relacionales sobre **la organización de los datos complejos e**s decir datos de texto e imagen, mapas, datos en el rango de audio etc.

Con las Bases de Datos **Objeto-Relacional** se amplía el modelo relacional destacando las siguientes aportaciones:

- Una base de datos orientada a objetos es una base de datos que incorpora todos los conceptos importantes del paradigma de objetos:
  - ➔ Encapsulación: propiedad que permite ocultar información al resto de los objetos, impidiendo así accesos incorrectos o conflictos.
  - ➔ Herencia: propiedad a través de la cual los objetos heredan comportamiento
  - ➔ Polimorfismo - Propiedad de una operación mediante la cual puede ser aplicada a distintos tipos de objetos.

- Se genera la posibilidad de guardar objetos más complejos en una sola tabla con referencias a otras relaciones, con lo que se acerca más al paradigma de programación orientada a objetos.

- Se pueden crear nuevos tipos de datos que permitan construir aplicaciones complejas con una gran riqueza de dominios. Se soportan tipos complejos como: registros, conjuntos, referencias, listas, pilas, colas y vectores.

  Se pueden crear procedimientos almacenados y funciones que tengan un código en algún lenguaje de programación, como por ejemplo: SQL, Java, C, etc.
- Se pueden compartir varias librerías de clases ya existentes, esto es lo que conocemos como reusabilidad.

Como productos comerciales de bases de Datos Objeto-Relacional podemos destacar:

DB2 Universal Database de IBM (International Business Machines).
Universal Server de Informix (ahora de IBM),
INGRES II, de Computer Associates.
ORACLE de Oracle Corporation,
SyBASE
PostGreSQL. (código abierto)

Si uno quiere manejar los tipos de datos complejos en una base de datos tendrá que tener en cuenta el diseño de una base datos objeto-relacionales por encima de una base de datos relacional ya que estas se pueden adaptar mejor a estos tipo de datos.

3.3 ACCESO A BASE DE DATOS MEDIANTE CONECTORES

Para gestionar la información de las bases de datos hay unos estándares que facilitan a los programas informáticos manipular la información almacenada en ellas.

En Java existe un API basado en estos estándares que permite desarrollar aplicaciones, que se pueda acceder a bases de datos relacionales: JDBC (Java Database Connectivity, conectividad de bases de datos de Java).

La mayoría de las aplicaciones importantes de una empresa están respaldadas por una arquitectura normalizada y optimizada de bases de datos relacionales. Tradicionalmente, dichas aplicaciones están basadas en sentencias SQL con las cuales se gestionan todos los datos que manejan.

Un driver JDBC es un componente software que posibilita a una aplicación Java interaccionar con una base de datos.

El API JDBC define interfaces y clases para escribir aplicaciones de bases de datos en Java realizando conexiones de base de datos.

Mediante JDBC el programador puede enviar sentencias SQL, y PL/SQL a una base de datos relacional. JDBC permite embeber SQL dentro de código Java. (embeber significa insertar o incrustar código de un lenguaje dentro de otro lenguaje).
La ventaja de usar conectores JDBC son:

- Independencia de la base de datos: JDBC proporciona una capa de abstracción que permite a los desarrolladores conectarse a una variedad de sistemas de gestión de bases de datos relacionales (como MySQL, PostgreSQL, Oracle, SQL Server) utilizando un conjunto común de clases y métodos. Esto facilita la migración de una base de datos a otra sin cambiar el código de la aplicación.

- Rendimiento: JDBC generalmente ofrece un buen rendimiento en comparación con otras formas de acceso a bases de datos, especialmente cuando se utiliza correctamente. Los controladores JDBC optimizados pueden ayudar a mejorar aún más el rendimiento.

- Seguridad: JDBC permite la implementación de medidas de seguridad, como la autenticación y la autorización, para proteger el acceso a la base de datos.

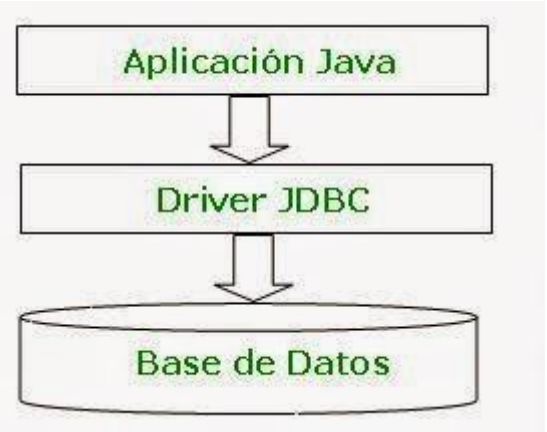
- Transacciones: JDBC es compatible con transacciones, lo que permite a las aplicaciones realizar operaciones de base de datos de manera atómica y

consistente. Esto es fundamental para mantener la integridad de los datos en aplicaciones empresariales.

- Facilidad de desarrollo: JDBC proporciona una interfaz de programación de aplicaciones clara y coherente que facilita el desarrollo de aplicaciones de bases de datos en Java. Los desarrolladores pueden utilizar lenguaje SQL estándar en combinación con JDBC para interactuar con la base de datos.

- Compatibilidad con ORM: JDBC se utiliza como base para muchas bibliotecas de mapeo objeto-relacional (ORM) populares, como Hibernate y JPA (Java Persistence API). Estas bibliotecas simplifican aún más la interacción con la base de datos al proporcionar un mapeo entre objetos Java y tablas de bases de datos.

- Control total: JDBC proporciona a los desarrolladores un mayor control sobre las consultas y operaciones de base de datos, lo que puede ser importante en situaciones en las que se requiere un ajuste fino de las consultas SQL o un acceso directo a características específicas de la base de datos.



3.4 MAPEO OBJETO RELACIONAL (ORM)

El mapeo objeto-relacional (ORM, por sus siglas en inglés, Object-Relational Mapping) es una técnica de programación que permite a los desarrolladores de software trabajar con objetos en un lenguaje de programación orientado a objetos (como Java, Python o C#) y mapear automáticamente esos objetos a tablas en una base de datos relacional.
El objetivo principal del ORM es facilitar y simplificar la interacción entre el código de la aplicación y la base de datos, eliminando la necesidad de escribir manualmente consultas SQL y realizar operaciones de mapeo de datos.

Ejemplo:
SIN ORM

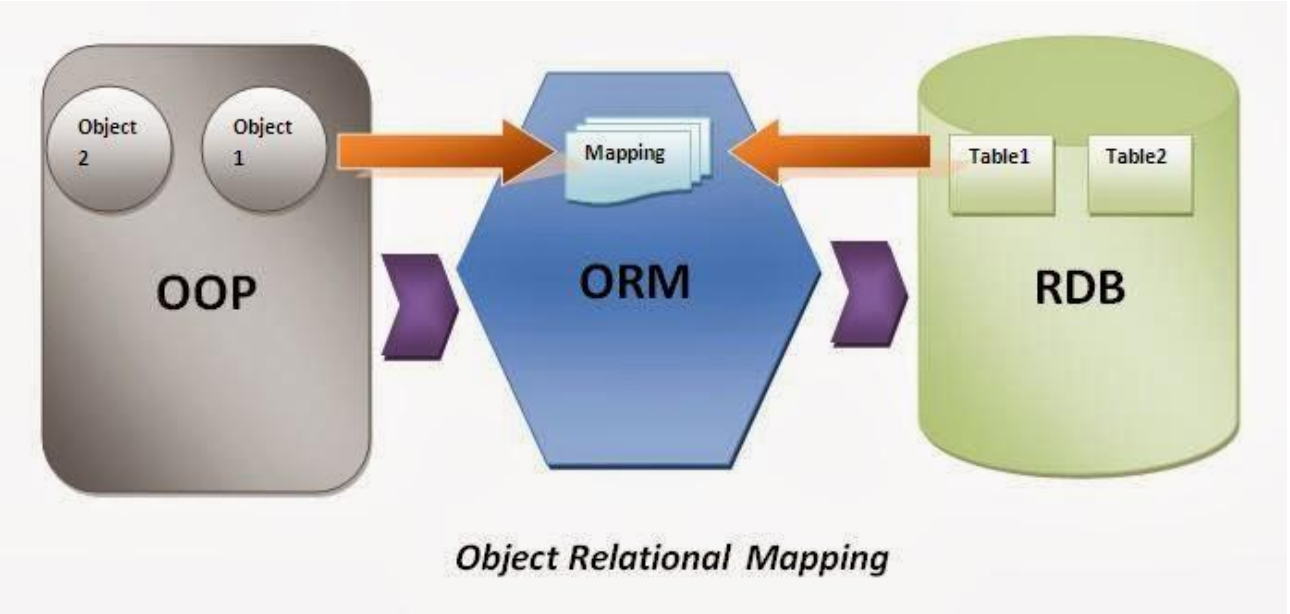
```
String query = "INSERT INTO clientes (id,nombre,email,pais) VALUES (@id, @nombre,
@email, @pais)";

command.Parameters.AddWithValue("@id","1")
command.Parameters.AddWithValue("@nombre","nombre")
command.Parameters.AddWithValue("@email","email")
command.Parameters.AddWithValue("@pais","pais")
command.ExecuteNonQuery();
```

CON ORM

```
var cliente = new Cliente();
cliente.Id = "1";
cliente.Nombre = "nombre";
cliente.Email = "email";
cliente.Pais = "pais";
session.Save(customer);
```

Un ORM te permite convertir los datos de tus objetos en un formato correcto para poder guardar la información en una base de datos (mapeo) creándose una base de datos virtual donde los datos que se encuentran en nuestra aplicación, quedan vinculados a la base de datos (persistencia).

El ORM realiza el mapeo entre las clases y objetos en el código de la aplicación y las tablas y registros en la base de datos. Esto significa que una clase se asigna a una tabla y sus atributos se asignan a columnas de esa tabla.



Cuando se trabaja con programación orientada a objetos y con bases de datos relacionales, es fácil observar que estos son dos paradigmas diferentes. El modelo relacional trata con relaciones y conjuntos, es de naturaleza matemática. Por el contrario, el paradigma orientado a objetos trata con objetos, atributos y asociaciones de unos con otros.

Por ello, para ahorrar trabajo al programador, se puede utilizar un **framework** que se encargue de realizar estas tareas de modo transparente, de modo que el programador no tenga por qué usar JDBC ni SQL y la gestión del acceso a base de datos esté centralizada en un componente único permitiendo su reutilización.

3.5 Bases de datos XML.

Debido a las limitaciones de la tecnología de bases de datos relacionales están surgiendo nuevas clases de bases de datos como son las bases de datos XML.

¿Qué es una Base de Datos XML?
Una base de datos en XML es un método de almacenamiento de información que permite albergar datos en formato XML. Estas bases de datos almacenan los datos estructurados como XML sin la necesidad de traducir los datos a una estructura relacional o de objeto.
Las bases de datos en XML tienen una serie de características que las diferencian del resto:

- Los datos se almacenan en archivos XML, que son documentos de texto plano que siguen las reglas de marcado de XML. Cada documento XML contiene datos organizados en una estructura específica, definida por un esquema o DTD (Document Type Definition) o un esquema XSD (XML Schema Definition).

- Consultas y manipulación: Para acceder y manipular los datos en una base de datos XML, se utilizan consultas XPath o XQuery. Estos lenguajes permiten buscar, recuperar y modificar datos dentro de los documentos XML.
- Las bases de datos XML se utilizan en una variedad de aplicaciones, como la gestión de contenido web, la representación de datos en sistemas de intercambio de información, el almacenamiento de configuraciones de aplicaciones y la representación de datos científicos.

  Podemos distinguir entre:

  XML Base de datos activada (XML- enabled)

  Se trata de bases de datos relacionales, en las que la información se almacenan en tablas. Las tablas se dividen en filas, que contienen los registros, y columnas, que contienen los campos.

  Una base de datos relacional XML-enabled permite obtener los resultados de las consultas en formato XML, de ahí que se enmarquen dentro de las denominadas XML-enabled database.

  Ejemplos

```
INSERT INTO Empleados (ID, Nombre, DatosHabilidades)
VALUES (1, 'Juan Pérez', '<Habilidades>
 <Habilidad>Programación</Habilidad>
<Habilidad>Base de Datos</Habilidad>
<Habilidad>Desarrollo Web</Habilidad>
</Habilidades>');

SELECT Nombre, DatosHabilidades.value('(Habilidades/Habilidad)[1]',
'VARCHAR(50)') AS PrimeraHabilidad
FROM Empleados
WHERE ID = 1;
```

  Aunque SQL Server es principalmente una base de datos relacional, proporciona herramientas y funciones para trabajar con datos XML dentro de una base de datos relacional, lo que puede ser útil en situaciones donde necesitas gestionar tanto datos relacionales como datos XML en un solo sistema de gestión de bases de datos.

  Base de datos XML nativa (NXD)
  Al contrario que las bases de datos relacionales, este tipo de base de datos en XML nativa no posee campos ni tablas, sino que almacena documentos XML. Es por ello que también se les denomina con frecuencia bases de datos centradas en documentos.

Ejemplo de bases de datos en XML

Documento xml

```
<libro>
        <titulo>El Gran Gatsby</titulo>
       <autor>F. Scott Fitzgerald</autor>
       <publicacion>1925</publicacion>
</libro>
```

Para almacenar este documento (xquery):

```
query version "3.1";

let $nuevoLibro :=
 <libro>
<titulo>El Gran Gatsby</titulo>
<autor>F. Scott Fitzgerald</autor>
<publicacion>1925</publicacion>
</libro>

let $ruta := '/coleccion-de-libros/libro.xml'

return
xmldb:store($ruta, $nuevoLibro)
```

En este ejemplo, $nuevoLibro representa el nuevo libro que deseas agregar y $ruta es la ubicación donde deseas almacenar el documento XML en la base de datos.

Luego, puedes realizar consultas XQuery para recuperar información de la base de datos.

```
xquery version "3.1";
for $libro in /coleccion-de-libros/libro
where $libro/publicacion > 2000
return $libro
```

4 DESARROLLO DE COMPONENTES

La expansión, en los últimos años, de Internet y el comercio electrónico ha llevado a la necesidad de adoptar nuevas tecnologías y nuevos requisitos de desarrollo en las aplicaciones informáticas.
Los requisitos de las aplicaciones informáticas son cada vez más complejos. Por ello, aparecieron las técnicas orientadas a objetos, para, entre otras cosas, intentar programar de manera que el código que se desarrolle pueda ser **reutilizable.**

Definición de componente.

Un componente es una unidad de software que realiza una función bien definida y posee una interfaz bien definida.

Un componente software posee interfaces especificadas contractualmente, pudiendo ser desplegado independientemente y puede interaccionar con otros componentes para formar un sistema.

Un interfaz es un punto de acceso a los componentes: permite a los clientes acceder a los servicios proporcionados por un componente.

La idea de dividir u organizar en trozos el software, o sea, dividirlo en componentes surge para reducir la complejidad del software. Así se permite la reutilización y se acelera el proceso de ensamblaje del software.

Los creadores de componentes pueden especializarse creando objetos cada vez más complejos y de mayor calidad.

El software se hace cada vez más rápido, de mejor calidad y a menor coste incluso de mantenimiento.

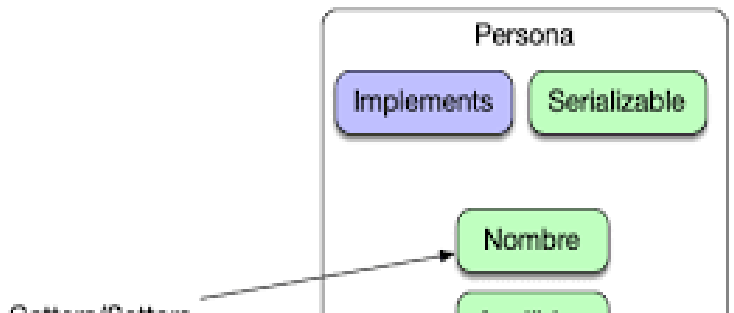
En el entorno Java, contamos con los **JavaBeans.**

JavaBeans.

La industria electrónica como en otras industrias se está acostumbrado a utilizar componentes para construir placas, tarjetas, etc. En el campo del software la idea es la misma. Se puede crear un interfaz de usuario en un programa Java en base a componentes: paneles, botones, etiquetas, caja de listas, barras de desplazamiento, diálogos, menús, etc.
Un JavaBean es un componente software reutilizable basado en la especificación JavaBean de Sun (ahora Oracle) que se puede manipular visualmente con una herramienta de desarrollo.

Alguna de las propiedades que presenta un JavaBean son:

- Portabilidad: uno de los principales objetivos de los JavaBeans es proporcionar una arquitectura neutral de componentes, es decir, que los beans puedan utilizarse en otras plataformas y entornos.
- Reusabilidad: son componentes reutilizables, la filosofía es que estos componentes pueden usarse como bloques en la construcción de aplicaciones complejas.
- Persistencia: un bean puede guardar su estado y recuperarlo posteriormente. Esta capacidad se logra mediante la serialización. Cuando un ejemplar de bean se serializa se convierte en un flujo de datos que se almacenará en algún sitio, probablemente en un fichero. Cualquier applet, aplicación o herramienta que utilice el bean puede restaurarlo mediante la deserialización.
- Comunicación entre eventos: Los eventos constituyen un mecanismo de notificación entre objeto fuente y objeto(s) receptor(es). Las herramientas de desarrollo pueden examinar un bean para determinar qué eventos puede enviar y cuáles puede recibir.



FIN