

Introducción al acceso a datos.

Caso práctico



Ada es una programadora experta en BK Programación, empresa que desarrolla software de diferente tipo y para diferentes plataformas, suele trabajar como supervisora en casi todos los proyectos que acomete la empresa.

María es técnica superior en Administración de Sistemas Informáticos. Suele dedicarse a mantener páginas web y a instalar servidores.

Juan es técnico superior en Desarrollo de Aplicaciones Informáticas, hace cuatro años que terminó el ciclo, y siente que necesita actualizarse en algunos aspectos. Fundamentalmente, se dedica a desarrollar aplicaciones web e instalar servicios de Internet.

La empresa también cuenta a partir de ahora con dos estudiantes: **Ana** y **Antonio**.

Ana, que es amiga de Juan y de María, en cuanto se enteró de la existencia del nuevo ciclo de Desarrollo de Aplicaciones Multiplataforma, se apuntó a estudiarlo para tener más posibilidades de trabajo en la empresa.

Antonio sabía algo de informática, pero poco, conocimientos básicos: manejaba paquetes ofimáticos y le gusta jugar con el ordenador. Su amiga **Ada** le propuso trabajar en la empresa, pero le comentó que tendría que estudiar, pues necesitaba tener más conocimientos que los que posee.

Ana está cursando el módulo de Formación en Centros de Trabajo (FCT), por lo que está aplicando lo que ha estudiado en el ciclo.

Una cosa que le llama la atención a **Ana** es que está constatando lo que sus tutores laborales, **María** y **Juan**, le comentaron. Justo antes de empezar el módulo de FCT, les dijeron que dependiendo de las necesidades de las aplicaciones informáticas, éstas, utilizarán un acceso a datos particular. Por eso, en una misma empresa u organismo, pueden utilizar un modo de guardar los datos para un cometido, y utilizar otro tipo, o estrategia, para guardar los datos de otra aplicación.

1.- Introducción.

Iniciamos esta primera unidad del módulo **Acceso a datos**, en el que veremos la gran variedad de métodos de acceso a datos que tenemos en el panorama actual.

Los contenidos del módulo son eminentemente prácticos, y los ejemplos estarán basados en Java. Utilizaremos como entorno de programación NetBeans, por coherencia con otros módulos del ciclo y por ser ambos gratuitos. Además, NetBeans es un entorno muy potente y fácil de usar, si lo comparamos con otros entornos de desarrollo.



Pero, ¿a qué nos referimos cuando hablamos de acceso a datos en una aplicación informática?

Podemos afirmar que en la inmensa mayoría de aplicaciones informáticas se pueden diferenciar, a grandes rasgos, en dos partes:

- ✓ Por un lado, **el programa** propiamente dicho, que realiza las operaciones deseadas con los datos necesarios.
- ✓ Por otro lado, **los datos** con los que opera el programa. Esos datos pueden ser obtenidos por el programa mediante diversos métodos: leídos mediante teclado, escaneados, leídos de algún soporte de almacenamiento secundario, etc.

En la mayoría de los casos, cuando programamos, nos interesa que el programa guarde los datos que le hemos introducido, o los resultados que dicho programa haya obtenido, de manera que si el programa termina su ejecución, los datos no se pierdan y puedan ser recuperados posteriormente, es decir, **persistan**. Una forma tradicional de hacer esto es mediante la utilización de ficheros o de bases de datos que se guardarán en un dispositivo de memoria no volátil (normalmente un disco).

Te habrás dado cuenta de que el almacenamiento en memoria RAM, mediante variables o vectores, es temporal, los datos se pierden cuando el programa termina. Quizás te habrá pasado alguna vez que, debido a un apagón eléctrico, has perdido el trabajo que estabas haciendo, que todavía no habías grabado. Los datos que se guardan en almacenamiento secundario, como ficheros o bases de datos, se denominan datos persistentes, porque existen, o persisten más allá de la ejecución de la aplicación.

Ese almacenamiento secundario de datos que acabamos de mencionar, habitualmente suele consistir en una base de datos relacional, si bien, a veces, hay otros métodos de almacenamiento, y por tanto, métodos de acceso a esos datos. De conocer esos tipos de almacenamiento y cómo acceder a ellos es de lo que trata este módulo.

En esta unidad inicial, vas a ver una panorámica de los diversos métodos de **persistencia** que encontramos en el mercado.



Autoevaluación

Señala la opción correcta. Para almacenar datos de manera persistente no utilizaremos:

- Ficheros de texto.
- Bases de datos orientadas a objeto.
- Memoria RAM.

Solución:

1. Incorrecto. ¡Incorrecto! Los datos almacenados en ficheros no se pierden.
2. Incorrecto. ¡No es correcto! Los datos almacenados en bases de datos no se pierden.

3. **Correcto.** ¡Exacto! Los datos almacenados en la RAM se pierden al apagar la computadora.

Para saber más

En este módulo y en el ciclo en general nos decantamos por Java. Otra tecnología alternativa a Java es .NET. Si quieres saber sobre ella visita el siguiente enlace de la Universidad de Granada:

[.NET](#)

2.- Acceso a datos.

Caso práctico



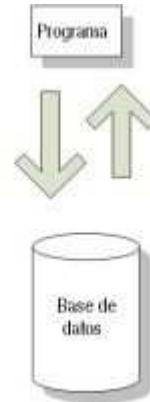
Antonio y Ana están hoy en la sede de BK Programación, pues han ido a recibir unos consejos y directrices de Ada, Juan y María. Tras la reunión, Antonio y Ana salen a tomar un café a un parque cercano. Compran un café y se lo llevan para tomar en un banco del parque, y mientras toman el café en la máquina, Antonio le pregunta a Ana: -¿Te has planteado la cantidad de estrategias de acceso a datos de datos que puede necesitar una empresa grande? -Ana se queda un momento pensativa y le dice a Antonio:-Pues la verdad es que no se me había ocurrido pensar en eso, pues tenía la idea de que una empresa sólo usaba un tipo de estrategia.

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

- ✓ Mediante **ficheros**.
- ✓ **Bases de datos**, que pueden ser:
 - ↳ Relacionales,
 - ↳ Orientadas a objetos,
 - ↳ Objeto-relacionales.
- ✓ **Mapeo objeto relacional (ORM)**.
- ✓ Bases de datos **XML** (eXtensible Markup Language).
- ✓ **Componentes**.

Al principio, en los primeros tiempos de la informática, los datos se guardaban en ficheros convencionales. Con el tiempo, y la experiencia de trabajar con dichos ficheros, se observaron los inconvenientes de los ficheros, y para intentar solucionar los inconvenientes que se observaron surgieron las bases de datos, que entre otras ventajas permitían:

- ✓ Eliminar el problema de la información redundante.
- ✓ Eliminar información inconsistente.
- ✓ Globalizar o centralizar la información.
- ✓ Garantizar el mantenimiento de la integridad en la información. Únicamente se almacena la información correcta.
- ✓ 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.



Autoevaluación

Uno de los objetivos de las bases de datos es que un cambio en los datos implique cambiar el programa de acceso a los mismos.

Verdadero. Falso.

Falso. La idea es la contraria a la afirmación, es decir, separar los datos de la aplicación en sí.

2.1.- Qué estrategia o método de acceso a datos usar.



Posteriormente, con la aparición y expansión de la programación orientada a objetos, empezaron a surgir las Bases de datos orientadas a objetos, y también se ampliaron algunas bases de datos relacionales, añadiéndoles extensiones de orientación a objetos.

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

Pues..., no hay una respuesta fácil para esas preguntas, no se puede afirmar que haya un método que sea el mejor de manera absoluta. Más bien, la cuestión es tener claro qué tipo de aplicación hay que construir y, según eso, estudiar qué tipo de sistema de almacenamiento será mejor usar: si una base de datos orientada a objetos, o una base de datos XML, etc.

sistema de almacenamiento será mejor usar: si una base de datos orientada a objetos, o una base de datos XML, etc.

Conociendo el funcionamiento de las diferentes alternativas podemos comparar sus prestaciones al problema de la persistencia concreto que se nos presente. Cada una de las tecnologías tiene su propio origen y filosofía para alcanzar el mismo fin y, por esta razón, no es fácil analizar sus ventajas y desventajas frente a las demás alternativas.

Por poner un ejemplo, lo más sencillo posible: 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.



Autoevaluación

Señala la opción correcta. La mejor opción para lograr la persistencia de los datos de una aplicación es:

- Utilizar una base de datos relacional.
- Emplear bases de datos orientadas a objeto.
- Usar ficheros de bajo nivel.
- Dependerá de las alternativas que haya y del sistema de información en estudio.

Solución:

1. **Incorrecto.** ¡No es correcto! Dependerá del sistema de información en cuestión.
2. **Incorrecto.** ¡Pues no! Dependerá del sistema de información en cuestión.
3. **Incorrecto.** ¡Incorrecto! Dependerá del sistema de información en cuestión.
4. **Correcto.** ¡Eso es! Dependerá del sistema de información en cuestión.

3.- Ficheros.

Caso práctico



Ada, María y Juan les explican a Ana y Antonio, que una empresa a la que hay que desarrollar un programa, tiene un par de aplicaciones antiguas. Esas aplicaciones son tan antiguas, que todavía usan ficheros planos para guardar los datos y, claro está, están pensando en evitar ese sistema de almacenamiento, y utilizar otro más adecuado a los tiempos que corren, evitando problemas como las redundancias e inconsistencias en los datos que se presentan

En las antiguas aplicaciones informáticas, antes de que surgieran las bases de datos, la información se guardaba en ficheros.

Así, por ejemplo, una aplicación que guardaba los datos de personas, almacenaba dichos datos en un fichero convencional cuyo contenido bien podía ser este:

Antonio Pérez Pérez 30 C/ Morales nº 11 Madrid Madrid

Feliciano Gómez Sander 25 C/ Terreros nº 121 Vitoria Vitoria

Arturo Bueno Hernández 46 C/ Cocoliso nº 43 Murcia Murcia

...

Esto tenía como efecto, que el programador de las aplicaciones que usaran ese fichero, tuviera que construir el programa conociendo detalladamente las posiciones de los datos, para saber desde qué posición hasta qué otra posición, se guardaba el nombre y apellidos, etc. Además, tendría que controlar si se guardan filas de datos duplicadas, y así un montón de inconvenientes. Por eso, cuando surgieron las bases de datos, se empezó a dejar de usar los ficheros convencionales.

Pero bien es cierto, que aún en las más modernas aplicaciones, a veces necesitamos un simple fichero para guardar información, como por ejemplo un fichero de configuración, o un fichero log. Es decir, no siempre nos hace falta una base de datos para almacenar la información.

En Java, como en otros lenguajes de programación, hay diversas clases para el manejo de ficheros, pues, como hemos dicho, a veces son muy útiles. Para guardar poca información, es mejor usarlos que usar otro método.

Nombre	Dirección	Ciudad	Provincia
Antonio Pérez Pérez	30 C/ Morales	nº 11	Madrid Madrid
Feliciano Gómez Sander	25 C/ Terreros	nº 121	Vitoria Vitoria
Arturo Bueno Hernández	46 C/ Cocoliso	nº 43	Murcia Murcia
...			

Reflexiona

Antiguamente, en el inicio de la informática, cuando no existían los discos duros, ni los ficheros, los datos y los programas se almacenaban en tarjetas perforadas. En esta entrada de blog tienes un ejemplo.

[Tarjetas perforadas](#)

En los años ochenta, los datos y programas se almacenaban en ficheros y éstos, en soportes como las cintas de casete, las cintas de música antiguas. Como anécdota, mira el vídeo promocional del ordenador ZX Spectrum+2, que incorporaba un lector de cassetes:

Spectrum+2

[Resumen textual alternativo](#)

3.1.- Uso ficheros en la actualidad.



Hay que tener en cuenta, que los ficheros en sí, para grabar la información del modo que poníamos como ejemplo anteriormente, en el ejemplo de las personas, ya no se usan. Pero, sí que se usan ficheros que guardan datos siguiendo un patrón o **estructura bien definida**, en otros métodos de almacenamiento, como por ejemplo en ficheros y en bases de datos XML.

De especial interés y uso cada vez más extendido, son los ficheros XML. Éstos son archivos de texto que por consiguiente no necesitan un software propietario para ser interpretados, como ocurre con la mayoría de los archivos binarios, y tienen normalmente la extensión xml.

También debemos tener en cuenta, que las bases de datos relativamente modernas, como son las bases de datos XML, guardan sus datos empleando ficheros xml.

Por eso, en muchas ocasiones se recurre a utilizar este tipo de soluciones, el uso de ficheros en vez de bases de datos, y en particular de ficheros XML 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.

Por esta razón se emplea XML en tecnologías de comunicación como, por ejemplo, en **WML** (lenguaje de formato inalámbrico) y **WAP** (protocolo de aplicaciones inalámbricas).

La fácil estructuración de la información en los ficheros XML ha permitido que surjan muchas librerías de conversión de la información almacenada a otros formatos como a **PDF**, texto, hojas de cálculo, etc. Hay muchos productos propietarios y de código abierto.



Autoevaluación

Señala la opción correcta. Respecto a los ficheros podemos asegurar que:

- Nunca es bueno utilizarlos.
- Los ficheros xml ya están en desuso.
- Depende de la aplicación, puede que en algún caso sean de utilidad.
- Fueron reemplazados por el formato pdf.

Solución:

1. **Incorrecto.** ¡No es correcto! A veces sí es la mejor opción.
2. **Incorrecto.** ¡Al contrario! Se usan cada vez más.
3. **Correcto.** ¡Así es! En ocasiones interesa usar ficheros y no bases de datos relacionales u otro tipo de estrategia de almacenamiento.
4. **Incorrecto.** ¡No! Al fin y al cabo, pdf es un formato de ficheros.

Para saber más

En el siguiente enlace tienes un tutorial de introducción al lenguaje XML:

[Tutorial XML](#)

Hemos mencionado el código abierto. En el siguiente enlace puedes ver una entrevista a Richard Stallman, impulsor del movimiento del software libre.

[Resumen textual alternativo](#)

4.- Bases de datos.

Caso práctico



Juan, después de la reunión que han tenido, sigue con su trabajo. Actualmente está trabajando en un proyecto, desarrollando unas mejoras, en una aplicación informática que gestiona una cadena de farmacias. En esas farmacias, guardan los datos en una base de datos relacional. Entre algunos de los datos, destacan los productos que venden, con sus precios, stock actual, precio de compra, etc. Juan opina tras estudiar la situación, que tal y como funciona el sistema, esa base de datos que usan está bien diseñada y no requerirá tocarla para nada, tan solo tendrá que conocer bien su estructura para programar las mejoras que necesitan.

Quizás hayas estudiado ya el módulo de bases de datos. Tanto si es así como si no, recordamos aquí qué es un sistema de bases de datos. Es:

- ✓ 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.



Citas para pensar

Elige por maestro aquel a quien admires, más por lo que en él vieras, que por lo que escucharas de sus labios.

Séneca.

4.1.- Introducción.

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. Esto supone un gran ahorro en los costes de programación, de forma que la modificación de la estructura de los datos no suponga un cambio en los programas y viceversa. Sin ella, el mantenimiento de la base de datos ocuparía el 50% de los recursos humanos dedicados al desarrollo de cualquier aplicación.
- ✓ Disminución de las **redundancias** y en consecuencia,
- ✓ Disminución de la posibilidad de que se produzca **inconsistencia** de datos.
- ✓ Mayor  **integridad de los datos**.
- ✓ Mayor **disponibilidad** de los datos.
- ✓ Mayor **seguridad** de los datos.
- ✓ Mayor **privacidad** de los datos.
- ✓ Mayor **eficiencia** en la recogida, codificación y entrada en el sistema.
- ✓ Lo que se suele denominar interfaz con el pasado y futuro: una base de datos debe estar abierta a reconocer información organizada físicamente por otro software.
- ✓ **Compartición** de los datos. Los datos deben poder ser accedidos por varios usuarios simultáneamente, teniendo previstos procedimientos para salvaguardar la integridad de los mismos.



Podemos afirmar generalizando, que se usa un sistema de ficheros convencional **cuando la cantidad de datos a guardar es tan reducida** que no justifica las desventajas del uso de los sistemas de bases de datos. Por ejemplo, para guardar los datos del resultado de la instalación de un programa, usamos un fichero de texto, no se guardan los datos en una base de datos.



Autoevaluación

Señala la opción correcta. El uso de bases de datos provoca que el usuario:

- Deba gestionar las posibles redundancias en los datos.
- Conozca la estructura de los ficheros índice de la base de datos.
- Tenga que implementar los procedimientos de seguridad en la base de datos.
- Ninguna es correcta.

Solución:

1. **Incorrecto.** ¡Incorrecto! No es tarea del usuario ocuparse de eso.
2. **Incorrecto.** ¡No es correcto! No es tarea del usuario.
3. **Incorrecto.** ¡Pues no! De eso se encarga el sistema gestor.
4. **Correcto.** ¡Eso es! El usuario no debe ocuparse de ninguno de esos aspectos.

4.2.- Bases de datos relacionales.

ID del cliente	Nombre cliente	Dirección cliente	Ciudad cliente	País cliente	Teléfono cliente	Fax cliente	E-mail cliente	Nº de teléfono
1	Alfreds Futterkiste	Mönchengladbach	Münster	Germany	02166 1234567	02166 1234567	ALFKI@FRANZSISKA.REWE.FR	02166 1234567
2	Extratrade Plc	London	London	United Kingdom	010 1234567	010 1234567	EXTRA@EXTRADE.UK	010 1234567
3	Island Trading	London	London	United Kingdom	010 1234567	010 1234567	IT@ISLAND-TRADING.CO.UK	010 1234567

Probablemente habrás cursado ya el módulo de bases de datos, si es así, e incluso si no lo es, quizás sepas que el **modelo de bases de datos relacional** fue propuesto en 1969 por Edgar Codd.

El propósito del modelo relacional es proporcionar un método declarativo para especificar datos y consultas. Así, en el diseño de la base de datos establecemos qué información contendrá dicha base de datos, luego recuperaremos la información que queramos, y dejamos al software del sistema gestor de la base de datos que se ocupe de: describir las estructuras de datos para almacenarlos, y gestionar los procedimientos de recuperación para obtener las consultas deseadas.

Las bases de datos relacionales son adecuadas para manejar grandes cantidades de datos, compartir datos entre programas, realizar búsquedas rápidas, etc. Pero tienen como desventaja fundamental que no presentan un buen modelo de las relaciones entre los datos, ya que todo se representa como tablas bidimensionales, o sea, en filas y columnas.

Podemos decir de las bases de datos relacionales que:

- ✓ Están muy extendidas.
- ✓ Son muy robustas.
- ✓ Permiten interoperabilidad entre aplicaciones.
- ✓ Permiten una forma de compartir datos entre aplicaciones.
- ✓ Son el común denominador de muchos sistemas y tecnologías. Una base de datos puede ser utilizada en programas realizados en Java, o en C++, etc.

Otros modelos tradicionales

Cuando se estudian las bases de datos relacionales, se suelen mencionar también los modelos **jerárquico** y **en red**. Algunos sistemas antiguos utilizan todavía estas arquitecturas, en centros de datos donde se manejan grandes volúmenes de información y la complejidad de los sistemas hace prohibitivo el coste que supondría migrar a sistemas que utilizaran otro modelo como el relacional.

El modelo relacional fue el primer modelo de bases de datos definido de manera formal. Posteriormente, se formularon modelos informales para describir bases de datos jerárquicas (el modelo jerárquico) y bases de datos en red (el modelo en red). Ambas, las bases de datos jerárquicas y en red existían antes que las bases de datos relacionales, **pero fueron descritas como modelo después** de que el modelo relacional fuera definido.

Para saber más

En esta presentación puedes ver algo más sobre los modelos en red y jerárquico.

 [Bases de datos. Modelos en red y modelo jerárquico \(1.58 MB\)](#)

4.3.- Bases de datos orientadas a objetos (I).

El origen de las Bases de datos orientadas a objetos (en adelante: BDOO) se debe básicamente a las siguientes razones:

- ✓ La existencia de problemas al representar cierta información y modelar ciertos aspectos del mundo real. Los modelos clásicos permiten representar gran cantidad de datos, pero las operaciones y representaciones que se pueden realizar sobre ellos son bastante simples.
- ✓ Pasar del modelo de objetos al modelo relacional, para almacenar la información, genera dificultades que en el caso de las BDOO no surgen, ya que el modelo es el mismo. Es decir, los datos de los programas escritos en lenguaje orientado a objetos se pueden almacenar directamente, sin conversión alguna, en las BDOO.



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.

Por ello, una característica general es que **el lenguaje de programación y el esquema de la base de datos utilizan las mismas definiciones de tipos**.

Para saber más

En este enlace está le manifiesto original de Malcolm Atkinson sobre las características que debe cumplir un sistema de bases de datos orientado a objetos.

 [Manifiesto original \(en inglés\)](#). (0.19 MB)

A continuación, se resume el manifiesto.

Resumen del manifiesto.

[Resumen textual alternativo](#)

4.3.1.- Bases de datos orientadas a objetos (II).

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 si se necesitan en los esquemas relacionales tabulares. Esto se debe a que un objeto puede recuperarse directamente sin una búsqueda, simplemente siguiendo punteros.

Cada vez más, las necesidades de las aplicaciones actuales con respecto a las bases de datos son:



- ✓ Soporte para objetos complejos y datos multimedia.
- ✓ Jerarquías de objetos o tipos y herencia.
- ✓ Gestión de versiones.
- ✓ Modelos extensibles mediante tipos de datos definidos por el usuario.
- ✓ Integración de los datos con sus procedimientos asociados.
- ✓ Manipulación navegacional (en vez de declarativa) y de conjunto de registros.
- ✓ Interconexión e interoperabilidad.

Como ejemplos de Sistemas Gestores de Bases de datos Orientados a Objetos podemos señalar:

- ✓ Jasmine

Base de datos Jasmine

- ✓ ObjectStore

Base de datos ObjectStore

- ✓ GemStone

Base de datos Gemstone

Para saber más

En este enlace puedes ver la definición de interoperabilidad, y también sobre la gestión de versiones, según la wikipedia.

[Definición de interoperabilidad](#)

[Gestión de versiones](#)



Autoevaluación

Señala las opciones correctas. Las bases de datos orientadas a objetos:

- Presentan conceptos de orientación a objetos como la herencia.
- Tienen el mismo tipo de problemas que las relacionales.

- No permiten la manipulación navegacional.
- Deben permitir el control de versiones.

[Mostrar Información](#)

Solución:

1. Selección correcta.
2. Selección incorrecta.
3. Selección incorrecta.
4. Selección correcta.

Retroalimentación:

Las bases de datos orientadas a objetos permiten usar conceptos de programación orientada a objetos y deben permitir la gestión de versiones.

4.4.- Comparativa entre bases de datos relacionales y orientadas a objetos.

En numerosos bancos de pruebas realizados para comparar los sistemas de bases de datos orientados a objetos (ODBMS) y los sistemas de bases de datos relacionales (RDBMS), se ha mostrado que los ODBMS pueden ser superiores para ciertos tipos de tareas.

La explicación a esto es que muchas operaciones se realizan utilizando interfaces navegacionales más que declarativos, y el acceso a datos navegacional es normalmente implementado muy eficientemente.

Un buen argumento a favor de las bases de datos orientadas a objetos es la transparencia (no ensucia la construcción de una aplicación). El desarrollador así se debe preocupar de los objetos de su aplicación, en lugar de cómo los debe almacenar y recuperar de un medio físico.



Podemos decir que las ventajas de un SGBDOO frente a las relacionales son:

- ✓ 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:
 - ↳ un objeto permite encapsular tanto un estado como un comportamiento
 - ↳ un objeto puede almacenar todas las relaciones que tenga con otros objetos
 - ↳ los objetos pueden agruparse para formar objetos complejos (herencia).
- ✓ **Extensibilidad**, debido a que:
 - ↳ Se pueden construir nuevos tipos de datos a partir de los ya existentes.
 - ↳ Podemos agrupar propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
 - ↳ Tenemos reusabilidad de clases, lo que repercute en una mayor facilidad de mantenimiento y un menor tiempo de desarrollo.
- ✓ Disposición de un **lenguaje de consulta más expresivo**. El **acceso navegacional** desde un objeto al siguiente es la forma más común de acceso a datos en un sistema gestor orientado a objetos. Mientras que **SQL** utiliza el acceso declarativo. El acceso navegacional es más adecuado para gestionar operaciones tales como consultas recursivas, etc.
- ✓ **Adaptación a aplicaciones** avanzadas de base de datos. Hay muchas áreas en las que las bases de datos relacionales no han tenido excesivo éxito, como es el caso de en sistemas de diseño CAD, CASE, sistemas multimedia, etc. en los que las capacidades de modelado de los SGBDOO han hecho que esos sistemas sí resulten efectivos para este tipo de aplicaciones.
- ✓ **Prestaciones**. Los sistemas gestores de bases de datos orientadas a objetos proporcionan mejoras significativas de rendimiento con respecto a los SGBD relationales. Aunque hay autores, que han argumentado que los bancos de prueba, usados en dichas pruebas, están dirigidos a aplicaciones de ingeniería donde los SGBDOO son más adecuados. También está demostrado, que los SGBDR tienen un rendimiento mejor que los SGBDOO en las aplicaciones tradicionales de bases de datos como el procesamiento de **transacciones** en línea (OLTP).
- ✓ **Reglas de acceso**. En las bases de datos relationales, a los atributos se accede y se modifican a través de operadores relationales predefinidos. En las orientadas a objetos se procede mediante las interfaces que se crean a tal efecto de las clases. Desde este punto de vista, los sistemas orientados a objetos dan independencia a cada objeto que el sistema relacional no permite.
- ✓ **Clave**. En el modelo relacional, las claves primarias generalmente tienen una forma representable en texto, sin embargo los objetos no necesitan una representación visible del identificador.

Para saber más

En esta dirección puedes ver más información sobre las bases de datos orientado a objetos.

 [Bases de datos orientadas a objetos.](#) (0.15 MB)

Reflexiona

Puedes ver un poco sobre la vida de Edgar F.Codd, el "padre" del modelo relacional en el siguiente enlace.

[Edgar F.Codd.](#)



Autoevaluación

Señala la opción correcta. Las bases de datos orientadas a objetos son más recomendables que las relacionales:

- Siempre.
- Cuando necesitamos la mayor cercanía al mundo real, la mayor capacidad de modelado, como en sistemas CAD/CAM.
- En aplicaciones de sistemas de control.
- En programas de control de stock.

Solución:

1. **Incorrecto.** No siempre es así.
2. **Correcto.** ¡Así es! Hay algunos casos como ese en los que las bases de datos orientadas a objetos son preferibles a las relacionales.
3. **Incorrecto.** ¡Incorrecto! Más bien en aplicaciones mulipedia, o CAD, etc.
4. **Incorrecto.** ¡No es correcto!

Para saber más

En la siguiente web puedes ver un curso introductorio sobre SQL.

[Curso de SQL](#)

4.4.1.- 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 un modelo de datos universal**. No hay ningún modelo de datos que esté universalmente aceptado para los SGBDOO y la mayoría de los modelos carecen de una base teórica. El modelo de objetos aún no tiene una teoría matemática coherente que le sirva de base.
- ✓ **Carencia de experiencia**. Al ser una tecnología relativamente nueva, todavía no se dispone del nivel de experiencia del que se 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 de facto. 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**.



4.5.- Bases de datos objeto-relacionales.

Entendemos Base de Datos Objeto Relacional (BDOR), una base de datos que ha evolucionado desde el modelo relacional a otro extendido que incorpora conceptos del paradigma orientado a objetos. Por tanto, un Sistema de Gestión Objeto-Relacional (SGBDOR) contiene ambas tecnologías: relacional y de objetos.

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 persona_t AS OBJECT (
    nif VARCHAR2(9),
    nombre VARCHAR2(30),
    direccion VARCHAR2(40),
    telefono VARCHAR2(15),
    fecha_nac DATE);
```

Por poner un ejemplo, si consideramos la base de datos Oracle, debido a los requerimientos de las nuevas aplicaciones, el sistema de gestión de bases de datos relacional desde versión 8i fue extendido con conceptos del modelo de bases de datos orientadas a objetos. De esta manera, aunque las estructuras de datos que se utilizan para almacenar la información siguen siendo tablas, los diseñadores pueden utilizar muchos de los mecanismos de orientación a objetos para definir y acceder a los datos. Se reconoce el concepto de objetos, de tal manera que un objeto tiene un tipo, se almacena en cierta fila de cierta tabla y tiene un identificador único (OID). Estos identificadores se pueden utilizar para referenciar a otros objetos y así representar relaciones de asociación y de agregación.



La ventaja de este tipo de base de datos es que los usuarios pueden pasar sus aplicaciones actuales sobre bases de datos relaciones este nuevo modelo sin tener que reescribirlos. Más tarde, se pueden ir adaptando las aplicaciones y bases de datos para que utilicen las funciones orientadas a objetos.

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

- ✓ Se aumentan la variedad en los tipos de datos,
 - ⇒ 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.
- ✓ Hay extensiones en el control de la Semántica de datos Objeto-Relacionales:
 - ⇒ 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

Como productos de código abierto, destacamos PostGreSQL.

Para saber más

 [Bases de datos Objeto-relacional](#) (0.33 MB)

5.- Acceso a bases de datos mediante conectores.

Caso práctico



Antonio, después de salir la reunión que tuvieron con Ada y los tutores, pensó que ya no se acordaba de una de las cosas que había comentado Ada, concretamente había dicho algo sobre JDBC. Antonio sabía perfectamente que lo había estudiado en el ciclo, pero ahora mismo no recordaba con total exactitud qué significaba ese término. Por ello, en cuanto llegara a casa, repasaría sus apuntes o lo buscaría por Internet, para calmar su inquietud.

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 al 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.

Este modelo continúa teniendo una gran importancia estratégica y es la base para el continuo crecimiento del mapeo Objeto-Relacional (O/R) y está asociado a los mecanismos de persistencia.

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

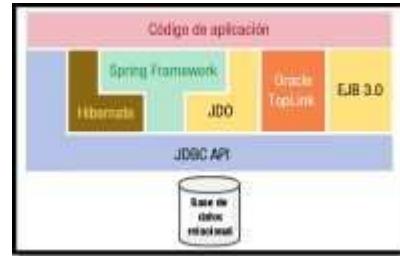
El API JDBC define interfaces y clases para escribir aplicaciones de bases de datos en Java realizando conexiones de base de datos.

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.

La ilustración muestra una representación de los diferentes mecanismos de mapeo O/R y cómo se relacionan con el código de la aplicación y con los recursos de datos relacionados. Se observa claramente la función crítica que desempeña el driver JDBC puesto que está situado en la base de cada uno de los marcos de trabajo.

La ventaja de usar conectores JDBC es que independiza de la base de datos que utilice.

No obstante hay un trabajo de traducción para mapear los campos devueltos por cada consulta a la colección de objetos correspondiente. Y hay que trabajar las sentencias de actualización, inserción y eliminación para cada uno de los campos. Esto constituye una razón para tratar de buscar alternativas menos costosas en tiempo de desarrollo.



¿Sabías que...?

El controlador de ODBC (Open Database Connectivity, Conectividad abierta de bases de datos) es la interfaz de programación de base de datos que utiliza Microsoft para tener acceso a distintas bases de datos relacionales en diversas plataformas. También existe un estándar que sirve de puente entre JDBC-ODBC en las versiones Solaris y Windows de la plataforma Java, para que se pueda utilizar  ODBC desde un programa Java.

6.- Mapeo objeto relacional (ORM).

Caso práctico



como parece.

María está trabajando con Ana en un proyecto que utiliza un acceso a datos de tipo mapeo objeto-relacional. Ana al principio se limita a ver y estudiar cuidadosamente lo que hace María, puesto que aunque estudió estos conceptos en el ciclo, ahora en una situación real, le impone bastante, y no quiere meter la pata. María por su parte intenta calmar a Ana, y le dice que no tenga miedo, y que siga atentamente las indicaciones que ella le va a dar, y verá como refrescando los conceptos, la cosa no es tan difícil

En los inicios de la programación, los programas se desarrollaban con la llamada programación imperativa. De hecho, y como sabes, la programación orientada a objetos surgió unos años después.

Con la expansión de la programación orientada a objetos, se da la circunstancia de que las bases de datos relacionales se siguen utilizando hoy en día de manera masiva, por lo que muchas aplicaciones se desarrollan con POO pero los datos, no se almacenan en bases de datos orientadas a objetos, sino en las bases de datos relacionales.



El sistema más extendido en las empresas hoy en día para guardar la información de sus aplicaciones es el uso de una base de datos relacional. Tradicionalmente, dichas aplicaciones están basadas en sentencias

SQL con las cuales se gestionan todos los datos que manejan. Este sistema es la base para el continuo crecimiento del mapeo Objeto-Relacional (O/R) y está asociado a los mecanismos de persistencia.

Cuando se programan sistemas orientados a objetos, utilizando una base de datos relacional, los programadores invierten gran cantidad de tiempo en desarrollar los objetos persistentes, o sea, convertir los objetos del lenguaje de programación a registros de la base de datos. Igualmente, también pasan bastante tiempo implementando la operación inversa, es decir, convirtiendo los registros en objetos.

El **mapeo objeto-relacional**(Object-Relational Mapping, o ORM) consisten en una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el sistema utilizado en una base de datos relacional.

Cuando se trabajan 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.

Cuando se requiere almacenar la información de los objetos utilizando una base de datos relacional se comprueba que hay un problema de compatibilidad entre estos dos paradigmas, el llamado **desfase objeto-relacional**.

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.



Autoevaluación

Señala las opciones correctas:

- El desfase objeto-relacional se refiere a la incompatibilidad que hay entre el paradigma relacional y el objeto-relacional.
- La programación orientada a objetos surgió antes que la programación imperativa.
- El modelo de bases de datos relacional todavía tiene un gran uso.
- La persistencia se define como la conversión de registros a objetos.

[Mostrar Información](#)

Solución:

1. Selección correcta.
2. Selección incorrecta.
3. Selección correcta.
4. Selección incorrecta.

Retroalimentación:

La programación orientada a objetos es posterior a la imperativa y desde luego que las bases de datos relacionales se siguen utilizando hoy en día.

6.1.- Capa de persistencia y framework de mapeo.



Capas de negocio, presentación y persistencia.

La **capa de persistencia** de una aplicación es la pieza que permite almacenar, recuperar, actualizar y eliminar el estado de los objetos que necesitan persistir en un sistema gestor de datos.

En el caso del mapeo objeto-relacional, un ORM es una capa que permite relacionar objetos con un modelo de datos relacional, ocultando todo el mecanismo de conexión al **motor de base de datos**, y también no teniendo que escribir las sentencias SQL necesarias para la gestión de los datos.

La **capa de persistencia** traduce entre los datos modelo de datos: desde objetos a registros y desde registros a objetos. Así, si el programa quiere grabar un objeto, entonces llama al **motor de persistencia**, el motor de persistencia traduce el objeto a registros y llama a la base de datos para que guarde estos registros.

De este modo el programa sólo ve que puede guardar y recuperar objetos, como si estuviera programado para una base de datos orientada a objetos. Y la base de datos sólo ve que guarda y recupera registros como si el programa estuviera dirigiéndose a ella de forma relacional.

Existen múltiples alternativas como desarrollador en Java cuando pretendas trabajar con mapeadores O/R. Hay tres comunidades que están implicadas en el mundo de la persistencia O/R de Java de forma activa:

- ✓ Organizaciones basadas en el estándar,
- ✓ Comunidades código abierto (open source) y
- ✓ Grupos comerciales.

Las comunidades open source incluyen importantes tecnologías, entre ellas Hibernate y el framework Spring.

Las alternativas más importantes basadas en el estándar, son EJB 3.0 y JDO.

Entre las implementaciones comerciales se puede resaltar TopLink.

Cada uno de los mecanismos de mapeo O/R tiene una dependencia particular en el conectar JDBC para poder comunicarse con la base de datos de una forma eficiente. Si el conectar JDBC que participa en la comunicación no es óptimo, la posible gran eficiencia de cualquier framework quedará debilitada. Por tanto, seleccionar el driver JDBC que mejor se adapte a la aplicación es esencial a la hora de construir un sistema eficiente en el que intervenga un mecanismo de mapeo O/R.

Debes conocer

No dejes de visitar el siguiente enlace de wikipedia al respecto de la persistencia.

Persistencia.



Autoevaluación

Señala las opciones correctas:

- No existen herramientas de código abierto relacionadas con el mapeo O/R.
- El motor de persistencia es un framework O/R.

- TopLink es un driver JDBC.
- Elegir un driver JDBC adecuado es fundamental para obtener un mapeo O/R eficiente.

[Mostrar Información](#)

Solución:

1. Selección correcta.
2. Selección incorrecta.
3. Selección incorrecta.
4. Selección correcta.

Retroalimentación:

Elegir un driver no adecuado hará que la velocidad de acceso a la base de datos sea bastante mejorable.

7.- Bases de datos XML.

Caso práctico

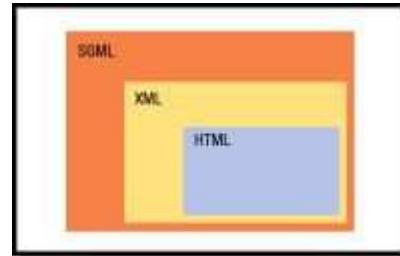


María, aunque hoy ha llegado algo cansada a casa, va a echar un rato en un proyecto de dispositivos móviles que está haciendo **con Juan**, antes de irse a la cama. En ese proyecto tienen que guardar información en un tipo de bases de datos que habitualmente no usaban en sus proyectos anteriores: las bases de datos XML.

Se define XML como: eXtensible Markup Language (lenguaje de marcado extensible). Es una especificación del World Wide Web Consortium (W3C).

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.

Estas bases de datos almacenan los datos estructurados como XML sin la necesidad de traducir los datos a una estructura relacional o de objeto.



Podemos distinguir entre:

- ✓ Bases de datos nativas XML. Consiste en un modelo lógico para documentos XML. El Sistema Gestor de Base de Datos correspondiente almacena y recupera documentos de acuerdo a dicho modelo.
 - ⇒ Productos ejemplo de esa filosofía son: eXist, Apache Xindice, Berkeley dbXML....
- ✓ Bases de datos compatibles con XML (XML-enabled): son bases de datos, generalmente objeto-relacionales, que admiten entradas de datos en forma de XML y proporcionan salidas en este mismo formato.
 - ⇒ Podemos citar productos como: Oracle, DB2 (Viper),...

Las bases de datos XML nativas permiten trabajar con XQL (eXtensible Query Language), el cuál sirve un propósito similar a SQL en una base de datos relacional.

El trabajo con bases de datos XML nativas involucra dos pasos básicos:

- ✓ Describir los datos mediante Definiciones de Tipos de Datos (Document Type Definitions, DTD) o esquemas XML y
- ✓ Definir un nuevo esquema de base de datos XML nativa o Mapa de Datos a usar para almacenar y obtener datos.

Para saber más

En este enlace puedes ver más sobre el lenguaje XQL.

[XQL](#)



Autoevaluación

Señala la opción correcta:

- XML es lo mismo que SQL pero para bases de datos orientadas a objetos.
- Oracle es una base de datos nativa XML.
- Una base de datos XML nativa puede trabajar con XQL.
- XQL es un lenguaje parecido a SQL, pero para bases de datos nativas XML.

[Mostrar Información](#)

Solución:

1. Selección incorrecta.
2. Selección incorrecta.
3. Selección correcta.
4. Selección correcta.

Retroalimentación:

Una base de datos XML nativa puede trabajar con XQL, que es un lenguaje parecido a SQL, para bases de datos nativas XML

8.- Desarrollo de componentes.

Caso práctico



Ada está pensando en un proyecto que han encargado a la empresa BK Programación recientemente. Es un proyecto ambicioso y bastante amplio. Piensa que en casi todos los proyectos en los que se embarcan, suelen utilizar partes de software parecidas, o a veces iguales que en otros proyectos. Por eso cree que podría aprovecharse esta cuestión e intentar programar componentes software que se puedan usar no en uno sino en muchos proyectos.

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.

Al principio de la informática, los ordenadores eran caros y la mano de obra de los programadores asequible. Con el paso del tiempo los ordenadores son baratos y la mano de obra de los programadores sigue siendo cara. Además, 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.



Para saber más

Aquí puedes ver información resumida sobre la programación con orientación a objetos.

[Tecnología orientada a objetos](#)

8.1.- 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.

La interoperabilidad entre componentes de distintos fabricantes:

- ✓ Aumenta la competencia,
- ✓ Reduce los costes y,
- ✓ Facilita la construcción de estándares.

Por tanto, así el software se hace cada vez más rápido, de mejor calidad y a menor coste incluso de mantenimiento.

Un componente software también debe especificar sus necesidades para funcionar correctamente, lo que se denominan las dependencias de contexto:

- ✓ Interfaces requeridas
- ✓ Entorno de ejecución: máquina necesaria, sistema operativo a utilizar, etc.

En el entorno Java, contamos con los JavaBeans.



Autoevaluación

Señala la opción correcta:

- Un componente software no necesita interfaces.
- Un componente software no puede interactuar con otro componente software.
- Quien vaya a usar un componente debe conocer el interface de dicho componente.
- Todas las afirmaciones anteriores son falsas.

Solución:

1. **Incorrecto.** ¡No es correcto! Los interfaces son imprescindibles en los componentes.
2. **Incorrecto.** ¡Incorrecto! Sí pueden interactuar.
3. **Correcto.** ¡Así es! Es necesario para poderlo utilizar correctamente.
4. **Incorrecto.** ¡Error! Sí hay alguna correcta.

8.2.- JavaBeans.

El **origen** de los JavaBeans lo podemos encontrar en un par de necesidades que Java tenía:

- ✓ Disponer de una tecnología de objetos y componentes reutilizables.
- ✓ Mejorar el proceso para crear interfaces de usuario, acercándose a la facilidad de uso que otros productos permitían como Visual Basic de Microsoft.



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.

Hay una serie de propiedades que presenta un JavaBean:

- ✓ **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.
- ✓ **Introspección:** los IDE reconocen ciertas pautas de diseño, nombres de las funciones miembros o métodos y definiciones de las clases, permitiendo a la herramienta de programación mirar dentro del bean y conocer sus propiedades y comportamiento.
- ✓ **Personalización:** en tiempo de diseño, con el IDE que se utilice, se pueden modificar las características de apariencia y comportamiento de un bean.
- ✓ **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.

Características de los JavaBeans.

[Resumen textual alternativo](#)

Para saber más

Aunque más adelante en el temario se profundizará sobre ellos, en el enlace siguiente puedes ver más sobre JavaBeans.

[JavaBeans](#)



Autoevaluación

Indica si la siguiente afirmación es verdadera o falsa:

Los JavaBean sólo se pueden usar en plataformas Windows.

Verdadero. Falso.

Falso. Evidentemente no, ya que son componente de Java y por tanto multiplataforma.

Anexo.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad

Recurso (1)	Datos del recurso (1)	Recurso (2)
	Autoría: Thomas Hawk. Licencia: CC-by-nc. Procedencia: http://www.flickr.com/photos/thomashawk/3318535957/	
	Autoría: nathalielaure. Licencia: CC-by. Procedencia: http://www.flickr.com/photos/nathalielaure/2787715668/	
	Autoría: Manuel Cernuda. Licencia: CC-by. Procedencia: http://www.flickr.com/photos/melkorcete/44959557/	
	Autoría: Extra Ketchup. Licencia: CC-by-sa. Procedencia: http://www.flickr.com/photos/extraketchup/749315946/	

