palasi.com
IT and management

Desarrollo de sistemas con metodología N-Tier, Visual Basic 6 y Access

FEPADE-ISEADE
23, 24 de noviembre y
1 de diciembre de 2001

Presentación del profesor

- Dr. Vicent-Ramon Palasí Lallana.
  - Licenciado en Informática por Universidad Politécnica de Cataluña (Barcelona, España)
  - Doctor en Ingeniería Informática por Universidad Jaume I (Castellón, España).
  - Diez años de experiencia profesional en el desarrollo de sistemas empresariales.
  - En la actualidad, Gerente General de Aurum Solutions: desarrollo de sistemas para empresas salvadoreñas y extranjeras.

Presentación de los participantes

- Nombre, empresa, a qué se dedican
- Tarjetas de identificación.
- Hojas de identificación. Grupos.

Programa del seminario (1)

- 1. Objetivos y metodología.
- 2. Introducción a la programación n-capas.
- 3. Descripción del modelo que se usará.
- 4. Diseño de la base de datos.
- 5. Active Data Objects. Recordsets desconectados.
- 6. Capa de datos.
- 7. Capa de dominio: Entidades.
- 8. Capa de dominio: Interfaces de VB.

Programa del seminario (y 2)

- 9. Capa de dominio: Clases de Soporte.
- 10. Capa de presentación.
- 11. Otros aspectos.
- 12. Conclusión.

Programa del seminario (1)

- 1. Objetivos y metodología.
- 2. Introducción a la programación n-capas.
- 3. Descripción del modelo que se usará.
- 4. Diseño de la base de datos.
- 5. Active Data Objects. Recordsets desconectados.
- 6. Capa de datos.
- 7. Capa de dominio: Entidades.
- 8. Capa de dominio: Interfaces de VB.

Mail: palasi@palasi.com Web: www.palasi.com

palasi.com
IT and management

Objetivos generales

- Asimilar los conceptos de la programación en N-capas.
- Aprender la forma de trasladar esos conceptos a un sistema programado en Visual Basic y ADO.
- Ser capaces de desarrollar un sistema de 3 capas de tamaño industrial.

Objetivos específicos

- Saber separar la lógica del programa en presentación, dominio y acceso a datos.
- Saber transmitir los datos entre capas.
- Utilizar ADO como una forma universal de acceso a datos.
- Saber programar para recordsets desconectados.
- Aprender el papel de las interfaces de VB.
- Saber implementar todos estos puntos en un ejemplo práctico.

Metodología utilizada

- Presentaciones en Powerpoint. Para transmitir los conceptos teóricos del seminario.
- Ejercicios. Para asimilar los conceptos en programas "de juguete".
- Laboratorio. Para programar los conceptos en un programa real.

Ventajas de esta metodología

- Combinación de teoría y práctica.
- Enfoque que va de lo abstracto a lo concreto.
- Fomenta la participación del alumno.

Programa del seminario (1)

- 1. Objetivos y metodología.
- 2. Introducción a la programación n-capas.
- 3. Descripción del modelo que se usará.
- 4. Diseño de la base de datos.
- 5. Active Data Objects. Recordsets desconectados.
- 6. Capa de datos.
- 7. Capa de dominio: Entidades.
- 8. Capa de dominio: Interfaces de VB.

Problemas del desarrollo de software

- El desarrollo de software se encuentra en un estado insatisfactorio y problemático (desde la conferencia de la OTAN de 1968 sobre Ingeniería del Software se denomina "crisis del software") .
- Las tareas que nos gustaría resolver mediante las computadoras son en la práctica:
  - demasiado difíciles de resolver.
  - suelen extenderse más allá del costo y el tiempo previsto.
  - es muy probable que contengan errores.

Mail: palasi@palasi.com Web: www.palasi.com

Problemas del desarrollo de software

- Los proyectos de programación de gran tamaño consumen 150% del tiempo previsto.
- El 25% de estos proyectos son cancelados.
- El 75% de los proyectos no cancelados:
  - O bien no funcionan como se quería.
  - O bien no se utilizan para nada.

Estadísticas sobre proyectos de software

Tamaño	Temprano	A tiempo	Retraso	Cancelados
1 PF	14.68%	83.16%	1.92%	0.25%
10 PF	11.08%	81.25%	5.67%	2.00%
100 PF	6.06%	74.77%	11.83%	7.33%
1000 PF	1.24%	60.76%	17.67%	20.33%
10000 PF	0.14%	28.03%	23.83%	48.00%
100,000PF	0.00 %	13.67%	21.33%	65.00%

- Fuente: Patterns of Software Systems Failure And Success, Capers Jones.

Programación con componentes reusables de software

- Una de las técnicas para mitigar los problemas del desarrollo de software.
- Se basa en la idea extendida en otras ingenierías que, para construir un producto, sólo deben ensamblarse piezas preconstruidas.
- La idea básica es:
  - existen componentes reusables
  - programar una aplicación consiste en ensamblarlos.

Programación orientada a objetos

- Es una de las técnicas para implementar programación con componentes reusables de software.
- Se basa, entre otros conceptos, en la encapsulación y la herencia.
- Estándar dominante actualmente de programación.
- Evoluciona de otras formas anteriores de programación, pero introduce una manera diferente de pensar (más parecida a los problemas de la vida real)

Un problema de la vida real

- Como enviar una remesa de EEUU a El Salvador.
- La solución es:
  - llegar a una empresa de mensajería privada, Remesa Express.
  - pedirles que quiero enviar el dinero a El Salvador.
- Remesa Express resolverá el problema con un método que no me interesa saber.
  - normalmente llamará la sucursal a El Salvador y les pedirá que entreguen el dinero.

Solución de problemas en la vida real

- En general:
  - se encuentra un **agente** que resuelva el problema.
  - se le pasa una **petición**.
  - el agente resuelve mi petición por un **método** que no tengo porque conocer (a menudo implica enviar un mensaje a otro agente)
- Por ejemplo, comprar un traje, pedir una pizza por teléfono...

Mail: palasi@palasi.com Web: www.palasi.com

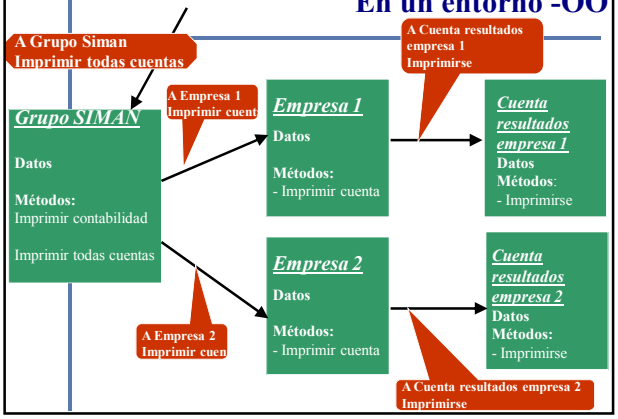
Solución de problemas en un entorno O-O

- La acción se inicia cuando
  - a un agente (objeto) se le pasa una petición (llamada mensaje o llamada a método).
- El objeto que recibe el mensaje resuelve la petición por un método
  - que no tenemos por que saber
  - que a menudo implica enviar otro mensaje a otro objeto.

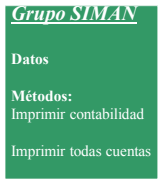
Un ejemplo de programación 0-0

- Supongamos que tenemos un sistema informático para el grupo de empresas SIMAN y queremos imprimir por pantalla la cuenta de resultado de cada empresa.

En un entorno -OO



Objeto

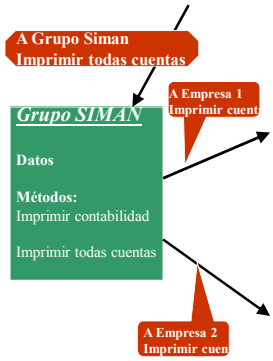


- Concepto principal de la programación O-O.
- Encapsulación. Contiene:
  - datos.
  - métodos para efectuar tareas sobre esos datos.
- Intenta modelar una entidad de la vida real.

Mensaje o llamada a método

- Es una petición a un objeto para que realice una tarea de las que sabe hacer este objeto.
- El objeto realiza la tarea siguiendo el método que tiene definido.

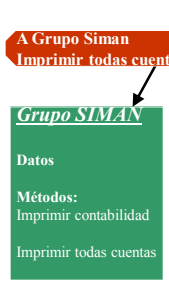
Método



- El objeto para realizar la tarea que ha pedido el mensaje tiene un método definido (a veces, enviar otro mensaje a otro objeto)
- Este método está oculto:
  - sólo lo conoce el objeto
  - no se conoce desde el exterior (excepto su nombre).

Mail: palasi@palasi.com Web: www.palasi.com

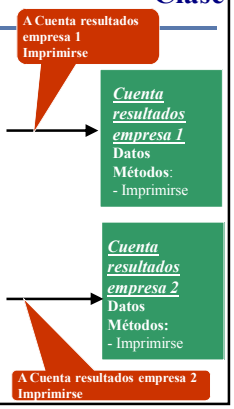
Ocultación de la implementación



- El objeto que ha llamado al método "Imprimir todas cuentas", no sabe cómo está implementado (sabe el QUÉ, pero no el CÓMO)
- Esta ocultación de la implementación es de lo más importante en O-O. Facilita la programación. Modelo de delegación.

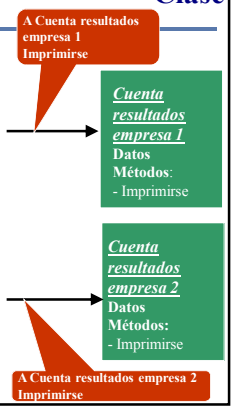
Clase

- Hay objetos que tienen el mismo tipo de datos y los mismos métodos para resolver una tarea.
- Esos métodos se agrupan en clases que son conjuntos de objetos con el mismo tipo de datos y los mismos métodos.



Clase

- Los objetos elementos de una clase se llaman instancias.
- En este caso, la clase sería la cuenta de resultados y sus instancias "cuenta de resultados 1 y 2"



Resumen de los conceptos de O-O hasta ahora

- Un programa O-O está formado por una serie de objetos que interactúan intercambiando mensajes (es decir, llamando métodos de otros objetos).
- Un objeto es un conjunto de datos y de métodos para realizar tareas con esos datos
- Los objetos con el mismo tipo de datos y los mismos métodos se agrupan en clases.

Otros conceptos de la Orientación a Objetos

- La orientación a objetos no es sólo lo que acabamos de explicar.
- Hay otros conceptos básicos, tales como herencia y polimorfismo.
- Pero estos conceptos no nos interesan por ahora. Hablaremos de ellos más adelante.

Visual Basic y O-O

- No es un lenguaje verdaderamente orientado a objetos pues le falta el concepto básico de herencia (aunque se puede simular).
- VB.NET sí contendrá herencia.
- Pero todos los conceptos explicados hasta ahora son aplicables a Visual Basic.
- VB suele considerarse un lenguaje de programación con componentes.

Mail: palasi@palasi.com Web: www.palasi.com

Pasando los conceptos a Visual Basic

- Para programar los objetos en Visual Basic, se deben programar las clases a las que pertenecen.
- En este caso, "grupo SIMAN" es una instancia de la clase grupos de empresas

Sintaxis de una clase en Visual Basic

- Debe estar contenida en un módulo de clase (p.ej., llamado "GrupoEmpresas").

```
Option Explicit
Public propiedad1 As tipo1
...
Public propiedadn As tipon

Public Function Metodo1(..) As tipo
End Public
...
Public Sub Metodon(...)
End Sub
```

Propiedades (datos)

Métodos (Sub o Func)

Pasando los conceptos a Visual Basic


```
Option Explicit
Public Empresa1 As  Empresa
Public Empresa2 As Empresa
…
Public Sub ImprimirContabilidad()
End Sub

Public Sub ImprimirTodasCuentas()
End Sub
...
```

Pasando los conceptos a Visual Basic

La clase Empresa ha sido definida en otra parte
Cada clase es un tipo de datos como integer.
La forma de definir un objeto de la clase X es declararlo como **nombreobjetoAs X**

```
Option Explicit
Public Empresa1 As
   Empresa
Public Empresa2 As
   Empresa
Public Sub
   ImprimirContabilidad()
End Sub

Public Sub
   ImprimirTodasCuentas()
End Sub
```

Pasando los conceptos a Visual Basic

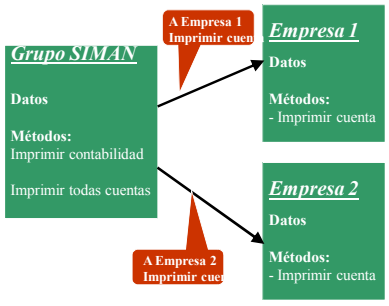
La sintaxis de un mensaje (o llamada a método) es la siguiente:
NomObjeto.NomMetodo (param1...paramn)

El mensaje que se muestra en la figura se escribiría en un Visual Basic:
Grupo.ImprimirTodasCuentas()

Los parámetros indican algunas informaciones útiles para la tarea a realizar.

La sintaxis es parecida a la de una llamada de procedimiento

Envío de un mensaje



Envío de un mensaje en Visual Basic

```
Option Explicit
Public Empresa1 As Empresa
Public Empresa2 As Empresa
...
Public Sub ImprimirTodasCuentas()
```

Clase GrupoEmpresas

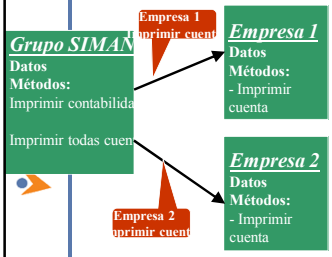
```
Option Explicit
...
Public Sub ImprimirCuenta()
...
```

Clase Empresa

Envío de un mensaje en Visual Basic

- Public Sub ImprimirTodasCuentas()
  Empresa1.ImprimirCuenta()
  Empresa2.ImprimirCuenta()
  End Sub
- Fíjense que este es el caso en que el método para resolver una tarea pedida por un mensaje implica enviar otros mensajes.

Envío de un mensaje en Visual Basic




```
Public Sub ImprimirTodasCuentas
    Empresa1.ImprimirCuenta()
    Empresa2.ImprimirCuenta()
End Sub
```

Crear un objeto en Visual Basic

- En Visual Basic.
  - Tenemos una clase clsX
  - Definimos una variable var1 As clsX
  - Lo que contiene la variable no es un objeto clsX, sino una referencia a un objeto de la clase clsX.
- Analogía de la dirección de una carta.
- El objeto debe ser creado y asignada su referencia a esta variable.
- Hasta ahora, habíamos supuesto que todos los objetos se habían creado

Cómo crear un nuevo objeto y asignar su referencia a 1 variable

- **Método 1: Dim var1 As New clsX**
  - La primera vez que se utilice la variable var1, se creará un nuevo objeto.
- **Método 2: Dim var1 As clsX [...]**
  Set var1 = New clsX
  - Con el Set New creamos el nuevo objeto y lo asignamos a una variable.
- **Método 3: Dim var1 As clsX [...]**
  Set var1 = CreateObject("clsX")
  - Con el CreateObject creamos el nuevo objeto y lo asignamos a una variable. Sólo si es un componente COM.

Características de los 3 métodos

- El método 2 es preferible al método 1 pues es más eficiente.
- El método 3 es preferible a los demás, pues podemos crear en una máquina un objeto de una clase que esté en otra máquina, lo que es importante en la programación 3 capas.
- En la práctica, se aconseja usar el método 3 (siempre que se asegure que se compila como COM). Ahora bien, en las transparencias, cuando no se suponen máquinas diferentes, a veces usamos el método 1 por su concisión.

Preguntas sobre orientación a objetos

Ejercicios

- *Dinàmica cartolines
- *Problema senzill de classes a Visual Basic.
- *Problema de quina classe va amb quina
- *Problema del deu errors

Ventajas de la orientación a objetos

- Organiza las cosas en el programa de forma similar como se organizan en el mundo real: objetos con comportamientos definidos.
- Más fácil de diseñar.
- Más fácil de programar.
- Más fácil de modificar y mantener.
- Mejor calidad del código.
- Mejor reusabilidad.
- Pero no es una “bala mágica”.

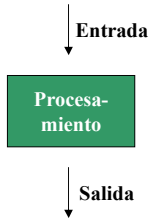
Problemas que siguen existiendo

- Cómo organizar la miríada de clases que se generan.
- Cómo hacer que un mismo código se aproveche para base de datos diferentes.
- Cómo hacer que un mismo código se aproveche para interfaces diferentes.
- Cómo adecuar la programación a modelos de computación distribuida.
- Cómo hacer más sencillo el mantenimiento.

Buscando una solución a estos problemas

- Cualquier solución que nos permita solucionar o mejorar varios de estos problemas significará un aumento de la calidad y una reducción de costos de la programación.
- Para hallar una solución, nos replantearemos el problema de la programación desde el principio.

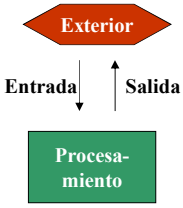
Esquema de procesamiento de datos



Mail: palasi@palasi.com Web: www.palasi.com

Esquema de procesamiento de datos

Esquema de procesamiento de datos

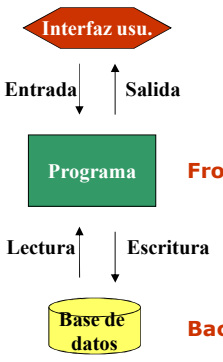


- Toda entrada y salida se efectúa hacia un exterior.

Esquema de procesamiento de datos

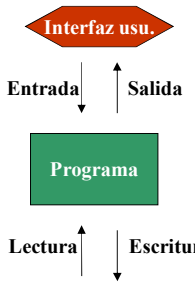
- En la informática, el procesamiento de datos se realiza con un programa o aplicación.
- La interfaz de usuario es la forma con la que el programa se comunica con el exterior (ventanas, pulsaciones de teclado)

Esquema de procesamiento de datos



- Los programas no triviales necesitan guardar datos en disco y acceder a ellos.

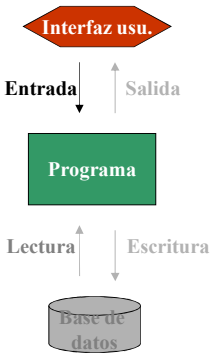
Esquema de procesamiento de datos



Tareas de un programa:

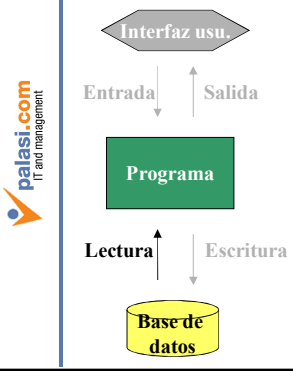
- 1. Controlar la entrada/salida desde la interfaz de usuario.
- 2. Hacer cálculos o procesamiento.
- 3. Controlar el acceso a la base de datos

Ejemplo: calcular la edad de un empleado (1)



1. Entrada. El programa recibe el nombre del empleado de un formulario por pantalla.

Ejemplo: calcular la edad de un empleado (2)

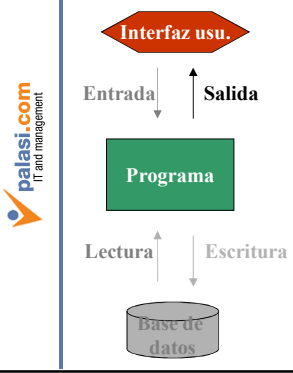


2. Lectura. El programa lee la fecha de nacimiento del empleado.

Ejemplo: calcular la edad de un empleado (3)

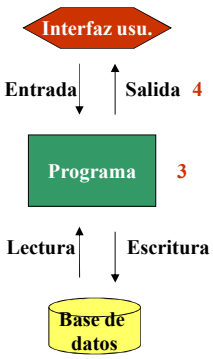
3. Cálculo. El programa lee la fecha de hoy y la compara con la fecha de nacimiento del empleado que ha recuperado de la base de datos consiguiendo así la edad.

Ejemplo: calcular la edad de un empleado (y 4)



4. Salida. El programa visualiza la edad del empleado por pantalla.

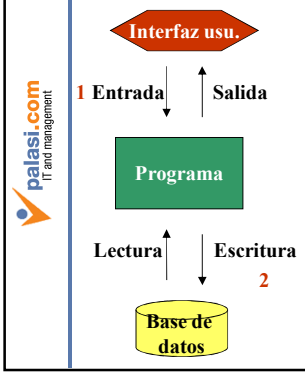
Ejemplo: calcular la edad de un empleado



Tareas a realizar

- 1. Entrada. Nombre empleado.
- 2. Lectura. Fecha de nacimiento.
- 3. Cálculo. Edad como diferencia.
- 4. Salida. Edad empleado

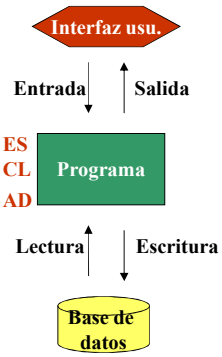
Ejemplo: dar de alta un empleado



Tareas a realizar

- 1. Entrada. Datos empleado.
- 2. Escritura. Se escribe los datos del empleado

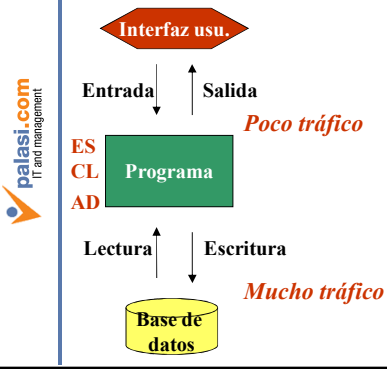
Tres tareas del programa



Cualquier programa tiene tres tipos de tareas:

- ES. Entrada y salida. El programa recibe y envía datos a la interfaz de usuario.
- CL. Cálculo. Se hacen manipulaciones con los datos.
- AD. Acceso a datos. El programa recibe y envíe

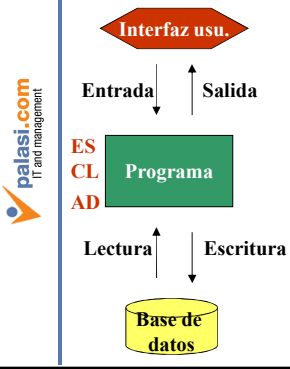
Tráfico desigual de datos



Como la interfaz de usuario puede mostrar pocos datos, el programa le envía pocos datos.

A veces, hay que recorrer toda una tabla para obtener los resultados que queremos.

¿Cómo se agrupan estas tareas?

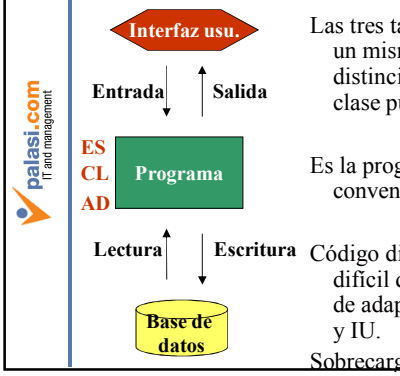


Como se agrupan los tres tipos de tareas en un programa:

- 1. Entrada y salida.
- 2. Cálculo.
- 3. Lectura y escritura.

- La forma de agruparse tiene repercusión sobre el tráfico en red, ya que el tráfico de datos no es homogéneo para las diferentes tareas.

Programación monolítica



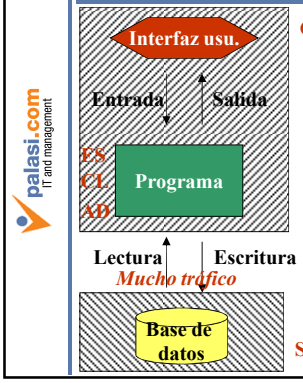
Las tres tareas se agrupan en un mismo programa sin distinción (por ej, una clase puede tener las tres).

Es la programación convencional.

Código difícil de mantener, difícil de reutilizar, difícil de adaptar a diferentes BD y IU.

Sobrecarga de red

Sobrecarga de red en programación monolítica



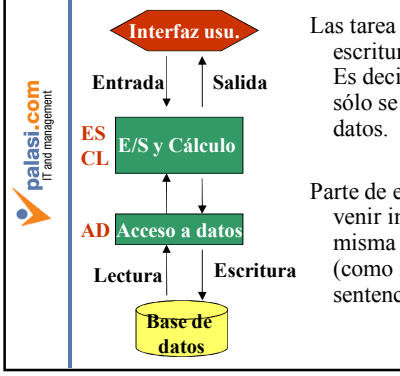
Cliente

La única forma de poner en red una arquitectura monolítica es colocando la BD en el servidor y el programa en cada uno de los clientes.

Dado que el tráfico de datos entre el programa y la BD es alto, hay una sobrecarga de red en este lo.

Servidor

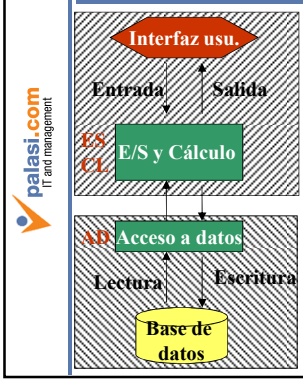
Programación en 2 capas (versión cliente-servidor)



Las tarea de lectura y escritura a datos se separa. Es decir, hay código que sólo se dedican al acceso a datos.

Parte de este código puede venir implementado en la misma base de datos (como implementación de sentencias SQL)

Red en programación en 2 capas (una posibilidad)



Cliente

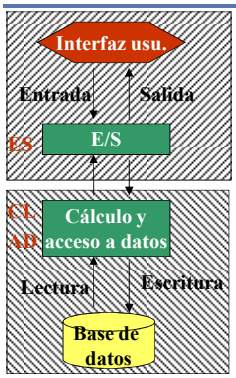
Como el tráfico de datos entre la base de datos y el acceso a datos se hace en una misma máquina o en máquinas cercanas, el tráfico en red disminuye.

Además, si cambiamos la base de datos, sólo cambiamos la programación de acceso a datos.

Servidor

Mail: palasi@palasi.com Web: www.palasi.com

Programación en 2 capas (versión Internet)



Cliente

La tarea de control d'E/S se separa. Es decir, hay código que sólo se dedica a entrada/salida.

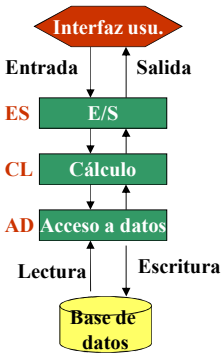
Poco tráfico.

Es adecuada para aplicaciones por Internet: en el cliente sólo hay el browser y en el servidor el resto.

Se puede combinar entre una aplicación de escritorio y de

Servidor :t

Programación en 3 capas



Cada una de las tres tareas de un programa se separa en clases diferentes.

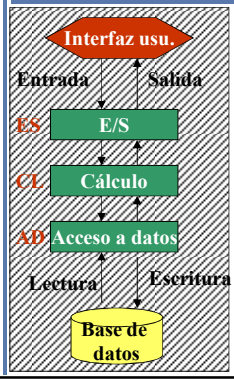
Combina las ventajas de las dos versiones de 2 capas:

Se puede cambiar la interfaz y la BD reutilizando el código.

Apropiada para Internet y escritorio.

Bajo tráfico de red.

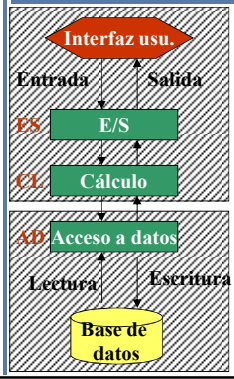
Diferentes posibilidades de distribución en red



Todo en 1 máquina

Conserva ventajas de reutilización de código y posibilidad de cambio de E/S y BD.

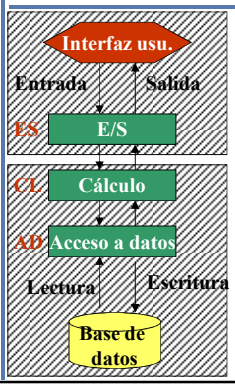
Diferentes posibilidades de distribución en red



Modelo cliente-servidor

La entrada/salida y el cálculo en el cliente. El acceso a datos en el servidor.

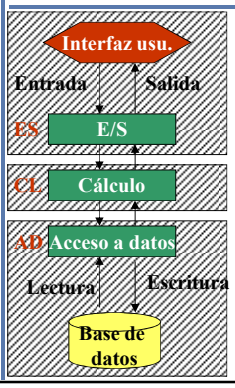
Diferentes posibilidades de distribución en red



Modelo "Internet"

La entrada/salida en el cliente. El cálculo y el acceso a datos en el servidor.

Diferentes posibilidades de distribución en red

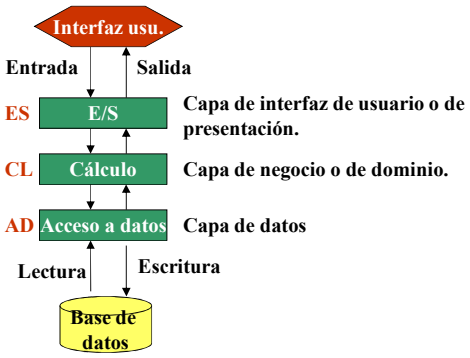


Cada capa en una máquina.

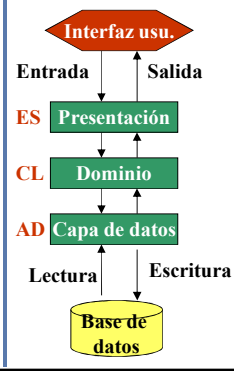
A la máquina que contiene el cálculo se le llama "Servidor de Aplicaciones". A la que contiene el acceso a datos, se le llama "Servidor de datos".

Mail: palasi@palasi.com Web: www.palasi.com

Un poco de terminología



Un poco de terminología



Definición de arquitectura en 3 capas

- Una arquitectura de software que divide la presentación (interfaz de usuario), lógica de la aplicación (o cálculo) y almacenamiento de datos en tres diferentes capas, llamadas respectivamente, capa de presentación, de dominio y de datos.
- La capa de presentación sólo comunica con la capa de dominio. La capa de dominio sólo comunica con la capa de datos.

Ventajas de la programación con 3 capas (1)

- El programa puede distribuirse en diferentes máquinas con la arquitectura que se desee.
- La capa de dominio y de presentación pueden colocarse tan cercanas como se pueda (idealmente en el mismo servidor). De esta manera, el tráfico en red disminuye.
- Puede programarse de forma que se pueda operar con diferentes BDs sólo cambiando la capa de datos.
- La programación puede servir para escritorio e Internet sólo cambiando la capa de presentación.

Ventajas de la programación con 3 capas (y 2)

- **Ventajas en el testing.** Como se tiene un interfaz definido entre el cliente y el servidor, es más fácil escribir un programa de prueba.
- **Ventajas en el balance de carga.** Si hay problemas de rendimiento, la capa de negocio puede ser movida a otros servidores en tiempo de ejecución.
- **Mejor gestión de versiones.** Es más fácil canviar un componente en el servidor que actualizar el programa en múltimples PCs.
- **Mejor actualización de sistemas.** Un sistema antiguo puede dotarse de una interfaz "wrapper" como la del objeto que queremos implementar.

Aplicaciones que se benefician de un diseño de 3 capas (1)

- Aplicaciones que necesitan exactitud en los cálculos. Los cálculos se efectúan en el servidor de aplicaciones (capa de negocio) y es posible equiparlo con procesadores matemáticos, etc.
- Aplicaciones que deben ser seguras (ej., bancos). Como se debe transferir al cliente tan poca información como sea posible, necesitamos una capa que nos ayude a eliminar la información no necesaria.

Mail: palasi@palasi.com Web: www.palasi.com

Aplicaciones que se benefician de un diseño de 3 capas (y 2)

- Aplicaciones que necesitan integridad de los datos. Como todo ello se realiza en el servidor, puede controlarse mejor la sincronización.
- Aplicaciones que deben ofrecer diferentes presentaciones. Estas presentaciones pueden ser programadas en la capa de presentación.
- Aplicaciones distribuibles por Internet.
- En general, cualquier aplicación que queremos que sea flexible y fácil de mantener.

Para resumir

- Las aplicaciones en 3 capas son:
  - Más predecibles.
  - Se desarrollan más eficientemente.
  - Más sencillas de probar
  - Más flexibles para cambios de base de datos y para actualizaciones.
  - Adecuadas para Internet y escritorio.

Ejercicios

- *Exemple Dinàmica cartolines 3-capes
- *Exercicis dinàmica cartolines.

Preguntas sobre 3 capas

Ejercicios

- *Es reparteix un codi font i es diu si és de tres capes o no.
- *Dir les modificacions necessàries per convertir l'anterior codi en tres capes.
- *Repetir els exercicis amb un altre exemple.

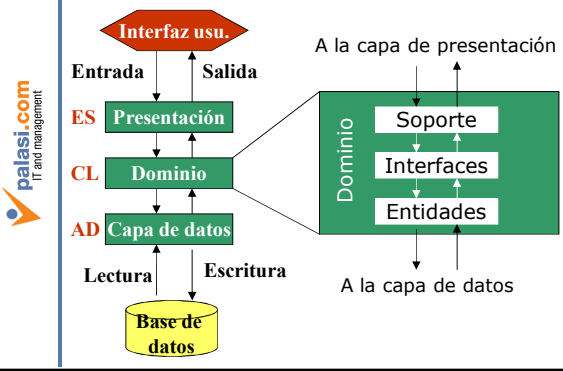
Laboratorio

- Implementar los ejercicios hasta ahora.

Programa del seminario (1)

- 1. Objetivos y metodología.
- 2. Introducción a la programación n-capas.
- 3. Descripción del modelo que se usará.
- 4. Diseño de la base de datos.
- 5. Active Data Objects. Recordsets desconectados.
- 6. Capa de datos.
- 7. Capa de dominio: Entidades.
- 8. Capa de dominio: Interfaces de VB.

Cada capa puede estar compuesta de varias subcapas

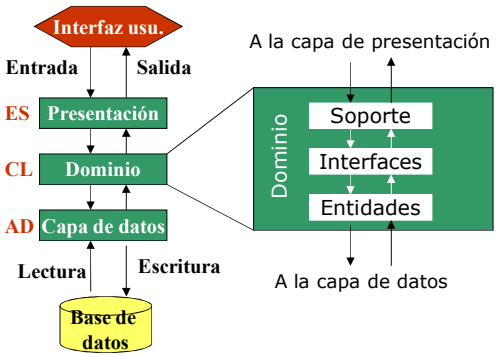


¿Qué significan los términos?

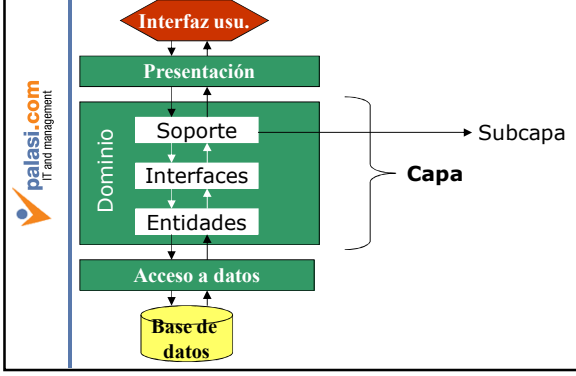
Definiciones de capa y subcapa

- Subcapa: Conjunto de clases que se definen en un nivel jerárquico de una aplicación. Las clases de una subcapa implementan su funcionalidad llamando a las clases de la subcapa inmediatamente inferior.
- Capas: Conjunto de subcapas que realizan una de las tres funciones básicas de una aplicación: presentación, lógica (o dominio) y acceso a datos.

Cada capa puede estar compuesta de varias subcapas

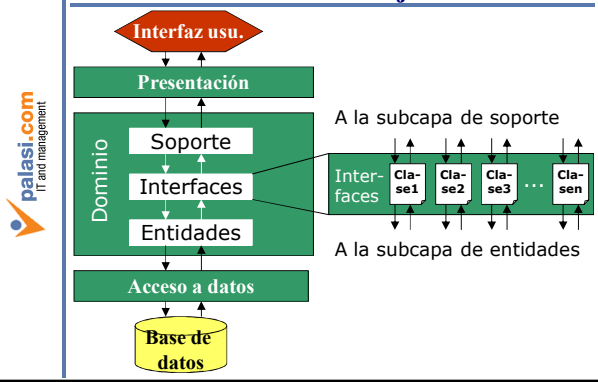


Cada capa puede estar compuesta de varias subcapas



Mail: palasi@palasi.com Web: www.palasi.com

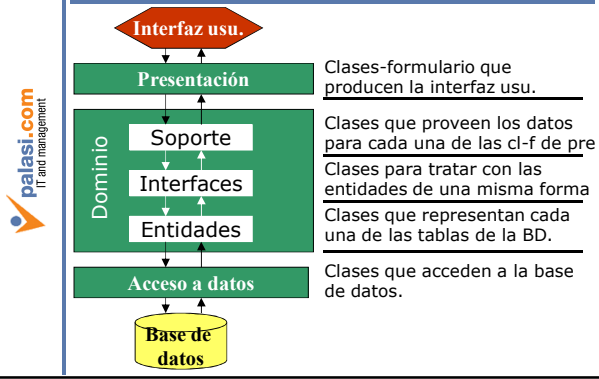
Cada subcapa es un conjunto de clases



Capas y subcapas

- Con la definición que hemos dado sólo pueden haber 3 capas: presentación, dominio y acceso a datos.
- Pero, ¿cuántas subcapas?
- Es una decisión de diseño:
  - Naturaleza de la aplicación.
  - Filosofía de diseño.
  - Etc.

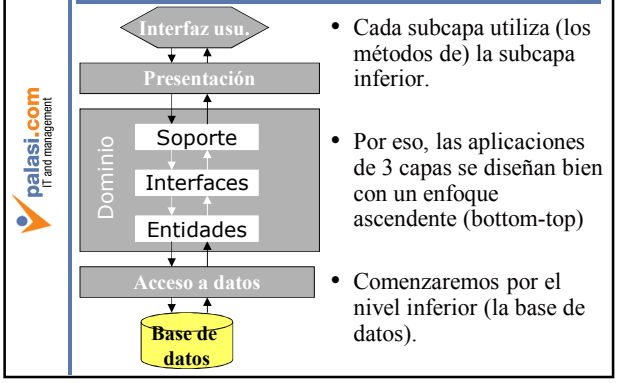
Nosotros usaremos el siguiente modelo



Programa del seminario (1)

- 1. Objetivos y metodología.
- 2. Introducción a la programación n-capas.
- 3. Descripción del modelo que se usará.
- 4. Diseño de la base de datos.
- 5. Active Data Objects. Recordsets desconectados.
- 6. Capa de datos.
- 7. Capa de dominio: Entidades.
- 8. Capa de dominio: Interfaces de VB.

Diseño de una aplicación de 3 capas



- Cada subcapa utiliza (los métodos de) la subcapa inferior.
- Por eso, las aplicaciones de 3 capas se diseñan bien con un enfoque ascendente (bottom-top)
- Comenzaremos por el nivel inferior (la base de datos).

Este no es un curso de bases de datos

- No se tiene tiempo para explicar métodos de diseño de bases de datos relacionales.
- Aún así, es necesario hablar brevemente del diseño de la BD si queremos entender los temas que vienen a continuación.

Mail: palasi@palasi.com Web: www.palasi.com

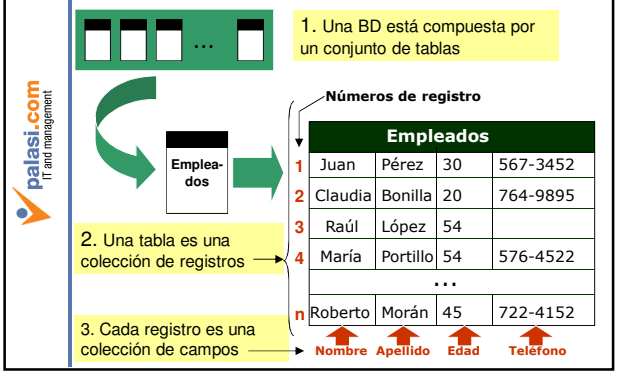
El modelo relacional

- Es el modelo en que se basan todas las bases de datos comerciales que hay actualmente en el mercado.
- Definido por E.F. Codd en 1970 pero no hubieron sistemas comerciales en 1977-78.
- Se han definido otros modelos posteriormente (por ejemplo, bases de datos orientados a objetos) pero no han tenido éxito.

El modelo relacional

- La colección de datos usada por el software se llama base de datos.
- La base de datos se divide en tablas, cada una de las cuales representa un conjunto de entidades de la vida real (así, tabla de empleados, de clientes, de facturas, etc.)
- Cada tabla es un conjunto de registros, cada uno de los cuales representa una entidad en la vida real (así, el registro que representa a Juan Pérez).
- Cada registro tiene una serie de datos llamados campos.

El modelo relacional



El modelo relacional es un modelo plano

- Los datos no tienen ninguna estructura.
- Esto es contrario a la programación orientada a objetos, donde los datos tienen estructura.
- Esto hace que la combinación de ambos no sea trivial.

Clave primaria en el modelo relacional

- Cada tabla debe tener un campo llamado clave primaria, que identifica unívocamente a cada uno de sus registros.

En este caso, la clave primaria es el campo "Cédula"

	Empleados				
1	566	Juan	Pérez	30	567-3452
2	099	Claudia	Bonilla	20	764-9895
3	567	Raúl	López	54	
4	927	María	Portillo	54	576-4522
		...			
n	456	Roberto	Morán	45	722-4152
	Cédula	Nombre	Apellido	Edad	Teléfono

Clave primaria en el modelo relacional

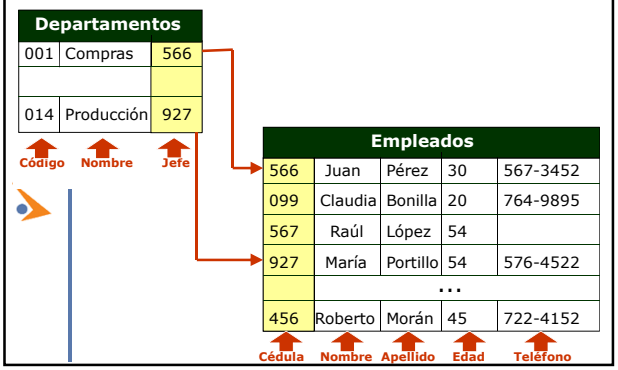
- Todos los campos de un registro deben depender de la clave primaria y sólo de ella.
- Asegurar esto es el propósito de las llamadas "formas normales"

Mail: palasi@palasi.com Web: www.palasi.com

Clave foránea

- En el modelo relacional, dos tablas se relacionan mediante una clave foránea: un campo de una tabla que es clave primaria en la otra tabla.

Clave foránea



Método sencillo de modelización de datos

- 1. Se escribe en una lista todos los datos que queremos modelizar. Se les da un nombre y un tipo.
- 2. Se identifican las diferentes tablas y se dividen los datos entre ellas.
- 3. Se identifica la clave primaria de cada una de las tablas.
- 4. Todo dato de una tabla debe depender de la clave primaria y sólo de la clave primaria. Si esto se cumple, ya hemos acabado.
- 5. En caso de que esto no se cumpla el dato debe ir en otra tabla. Debemos crear una nueva tabla, ligarla a la primera con una clave foránea y volver

Ejemplo de método de modelización de datos

- Programa de inscripción de asistentes a un congreso. Hay tres tipos de tarifas: estudiante, invitado y normal cuyos precios respectivos son 50, 100 y 200.
- Si el país de origen del asistente es centroamericano, el asistente llegará por autobús. En caso contrario, vendrá por avión y habrá que ir a recogerlo en el aeropuerto.
- De cada asistente nos interesa su nombre, apellidos, número de cédula, nacionalidad, qué tipo de tarifa, qué cantidad debe pagar, teléfono, dirección en el país, hotel en el que se aloja y si

1. Lista de todos los datos

- Nombre. (nombre). Texto(50)
- Apellidos (apellidos) Texto(50)
- Cédula (cedula) Numérico(20).
- Nacionalidad (pais) Texto(50).
- Tipo de tarifa (tipotarifa) Texto(10).
- Cantidad a pagar (importe) Numérico (3).
- Teléfono (telefono). Texto(7).
- Dirección en el país (direccion). Texto (150).
- Hotel (hotel). Texto (50).
- ¿Recoger en el aeropuerto? (avion) Lógico (1).

2. Identifica las tablas

- En principio, hay una tabla que parece clara: la tabla de asistentes.
- Como hipótesis de trabajo, suponemos que sólo hay esa tabla.

3. Identificar claves primarias

- Nombre. T(50)
- Apellidos T(50)
- Cedula N(20).
- Pais T(50).
- Tipotarifa T(10).
- Importe N(3).
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).
- Avion L(1).

- Debe identificar unívocamente al asistente.
- Nombre y apellidos identifican pero no unívocamente.
- No hay ningún campo que pueda ser clave primaria.

3. Identificar claves primarias

- Como no hay ningún campo que pueda ser clave primaria y necesitamos una clave primaria, debemos introducir un nuevo campo.
- Lo haremos de tipo autonumérico pues son adecuados para las claves.
- Le llamaremos "CodAsist" (en este caso, lo podemos aprovechar como código de inscripción).

3. Identificar claves primarias

- CodAsist. Autonum(6). CLAVE
- Nombre. T(50)
- Apellidos T(50)
- Cedula N(20).
- Pais T(50).
- Tipotarifa T(10).
- Importe N(3).
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).
- Avion L(1).

4. Los datos deben depender de la clave primaria y sólo de ella

- CodAsist. A(6). CLAVE. Identif. asistente
- Nombre. T(50). Depende de clave, del asistente.
- Apellidos T(50). Depende de clave.
- Cedula N(20). Depende de clave.
- Pais T(50). Depende de clave.
- Tipotarifa T(10). Depende de clave.
- Importe N(3). Depende de clave y de tipotarifa.
- Telefono T(7). Depende de clave.
- Direccion T(150). Depende de clave.
- Hotel. T(50). Depende de clave.
- Avion L(1). Depende de clave y de pais.

4. Los datos deben depender de la clave primaria y sólo de ella

- CodAsist. A(6). CLAVE.
- Nombre. T(50).
- Apellidos T(50).
- Cedula N(20).
- Pais T(50).
- Tipotarifa T(10).
- Importe N(3).
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

4.Hay campos que dependen de otros que no son la clave

- Tal como dice el método, separamos estos campos en una nueva tabla y la ligamos con una clave foránea.

5. Separando los datos en tablas

- CodAsist. A(6). CLAVE.
- Nombre. T(50).
- Apellidos T(50).
- Cedula N(20).
- Pais T(50).
- Tipotarifa T(10).
- Importe N(3).
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

5. Separando los datos en tablas

Asistentes
- CodAsist. A(6). Nombre. T(50).
- Apellidos T(50).
- Cedula N(20).
- <clave foránea tabla pais>
- <clave foránea tabla tipo de tarifa>
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

TiposTarifa
- TipoTarifa T(10)
- Importe N(3)

Paises
- Pais T(50)
- Avion L(1)

3. Identificar claves primarias

- TipoTarifa y Pais podrían ser claves primarias pues identifican unívocamente la tabla.
- Pero no es buena práctica que las claves primarias puedan ser introducidas por el usuario.

TiposTarifa
- TipoTarifa T(10)
- Importe N(3)

Paises
- Pais T(50)
- Avion L(1)

¿Por qué no claves primarias introducidas por el usuario?

- Se puede hacer pero no es buena práctica.
- Es difícil para el usuario asegurar la unicidad de la clave primaria.
- Los cambios de clave primaria pueden afectar otras tablas.
- Puede haber conflictos multiusuario.
- No se hace un uso totalmente eficiente de los recursos.
- Mejor hacer que las claves primarias las gestione el sistema.

3. Identificar claves primarias

- Colocaremos dos campos nuevos que no sean introducidos por el usuario para ser clave primaria.
- Es conveniente que sean campos autonuméricos

TiposTarifa
- CodTarifa A(3)
- TipoTarifa T(10)
- Importe N(3)

Paises
- CodPais A(3)
- Pais T(50)
- Avion L(1)

Claves

Determinando cuáles son las claves foráneas

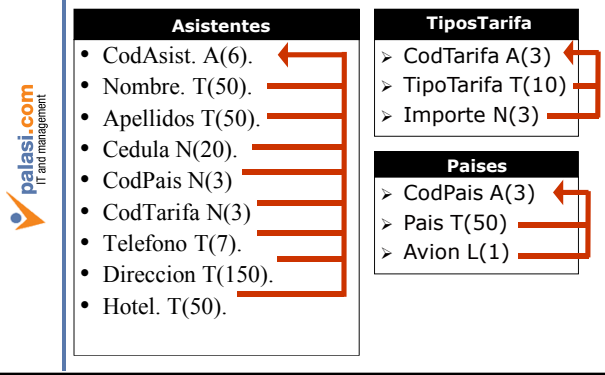
Asistentes
- CodAsist. A(6).
- Nombre. T(50).
- Apellidos T(50).
- Cedula N(20).
- CodPais N(3)
- CodTarifa N(3)
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

TiposTarifa
- CodTarifa A(3)
- TipoTarifa T(10)
- Importe N(3)

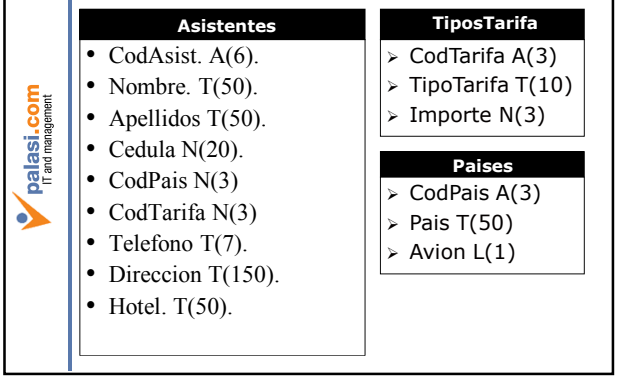
Paises
- CodPais A(3)
- Pais T(50)
- Avion L(1)

Mail: palasi@palasi.com Web: www.palasi.com

4. Los datos deben depender de la clave primaria y sólo de ella



Ya hemos acabado



Una observación

- Este ejercicio se ha realizado sobre el modelo relacional sin entrar en detalles específicos de una base de datos.
- Según sea la base de datos, puede ser necesario adaptar algunos aspectos menores del resultado
- Por ejemplo, en Access, no se puede especificar el número de dígitos de un campo numérico.

Preguntas sobre modelización de datos

Ejercicio de modelización de datos

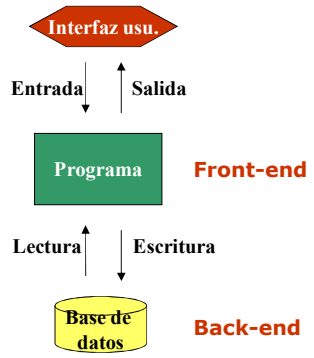
- Queremos implementar un sistema de facturación. Para cada factura, nos interesa el tipo de documento (comprobante, factura, factura de exportación, nota de crédito, nota de débito), la fecha, el número de documento, el nombre del cliente, el teléfono del cliente, el número de registro fiscal del cliente, sucursal, si es al crédito, el total de ventas exentas, el total de ventas gravadas y el tipo de IVA (que es 13% para las ventas nacionales y 0% para las exportaciones).

Programa del seminario (1)

- 1. Objetivos y metodología.
- 2. Introducción a la programación n-capas.
- 3. Descripción del modelo que se usará.
- 4. Diseño de la base de datos.
- 5. Active Data Objects. Recordsets desconectados.
- 6. Capa de datos.
- 7. Capa de dominio: Entidades.
- 8. Capa de dominio: Interfaces de VB.

Mail: palasi@palasi.com Web: www.palasi.com

Interfaz de acceso a datos



- Son un conjunto de clases que permiten al programa (o front-end) acceder a la base de datos (o back-end).

Interfaces de acceso a datos en Visual Basic

- En Visual Basic, ha habido diferentes interfaces de acceso a datos:
- DAO (Data Access Objects)
- RDO (Remote Data Objects)
- ADO (ActiveX Data Objects)
- ADO.NET (en versión beta)

¿Qué es ADO?

- ADO significa Objetos de Datos ActiveX (**A**ctiveX **D**ata **O**bjects)
- ADO es una interfaz de programación para acceder datos en una base de datos.
- ADO es un componente Active-X.
- ADO es un conjunto de clases.
- ADO es una tecnología Microsoft.

¿Qué es ADO?

- ADO significa Objetos de Datos ActiveX (**A**ctiveX **D**ata **O**bjects)
- ADO es una interfaz de programación para acceder datos en una base de datos.
- ADO es un componente Active-X.
- ADO es un conjunto de clases.
- ADO es una tecnología Microsoft.

Clases de ADO (1)

- Connection.
  - Representa una conexión a un origen de datos (pej, una base de datos).
- Recordset
  - Representa un conjunto de registros recuperado del origen de datos.
- Command
  - Representa una consulta que se ejecuta sobre un origen de datos.
- Error
  - Representa los errores de ADO.

Clases de ADO (y 2)

- Field
  - Representa información sobre una columna en un recordset.
- Parameter
  - Representa un parámetro usado en una consulta representado por Command.
- Property
  - Representa las propiedades del objeto ADO.

Mail: palasi@palasi.com Web: www.palasi.com

Clases de ADO que estudiaremos

- Connection.
- Recordset

La clase Connection

- Representa una conexión a un origen de datos. En nuestro caso, siempre será una base de datos.
- Propiedades más comunes:
  - Provider. Fija o devuelve el nombre del proveedor de datos (SGBD o similar).
  - Mode. Fija o devuelve los permisos de acceso del proveedor.
  - State. Devuelve un valor describiendo si la conexión está abierta o cerrada.
- Métodos más comunes.
  - Open. Abre una conexión.
  - Execute. Ejecuta una consulta o acción sobre el origen de datos.
  - Close. Cierra una conexión.

La clase Recordset (1)

- Representa un conjunto de registros recuperado del origen de datos.
- Propiedades más comunes:
  - BOF. Indica si la posición actual se encuentra antes del primer registro.
  - EOF. Indica si la posición actual se encuentra después del último registro.
  - Fields. Contiene todos los campos del Recordset.
  - RecordCount. Devuelve el número de registros que hay en el Recordset.
  - ActiveConnection y CursorLocation. Entre otras cosas, se utilizan para crear recordsets desconectados.

La clase Recordset (y 2)

- Métodos más comunes:
  - Open. Abre un Recordset
  - Close. Cierra un Recordset
  - AddNew. Crea un nuevo registro.
  - Delete. Borra el registro actual.
  - Update. Graba los cambios realizados sobre el registro actual.
  - MoveFirst. Se mueve al 1er. Registro.
  - MoveLast. Se mueve al último registro.
  - MoveNext. Se mueve al registro siguiente.
  - MovePrevious. Se mueve al registro anterior
  - Find. Se mueve a un registro que cumpla un

Recordsets desconectados (1)

- Los recordsets son la forma de acceder, p.ej., a una tabla desde Visual Basic.
- Tradicionalmente, los recordsets se encuentran permanentemente conectados a la BD. De esta forma, los cambios en el recordset se transmiten automáticamente a la BD.
- Ahora bien, esto no es adecuado para la programación con Internet, ya que no siempre existe una conexión continua con la BD.

Recordsets desconectados (y 2)

- La solución son los recordsets desconectados cuyo ciclo de vida es el siguiente.
- 1. Nos conectamos a la BD y creamos un recordset.
- 2. Nos desconectamos de la BD, creando así un recordset desconectado.
- 3. Realizamos los cambios en el recordset.
- 4. Nos volvemos a conectar para transmitir los cambios a la BD.
- A partir de ahora, sólo trataremos con recordsets

El modelo de programación ADO (1)

- 1. Haga una conexión a un origen de datos (Connection). Opcionalmente, comience una transacción.
- 2. Opcionalmente, puede crear un objeto Command para representar un comando SQL.
- 3. Opcionalmente, especifique columnas, tablas y valores dentro del comando SQL con parámetros variables (Parameter).
- 4. Ejecute el comando (Command, Connection o Recordset).
- 5. Si el comando regresa registros, almacénelos en un objeto de almacenamiento (Recordset).

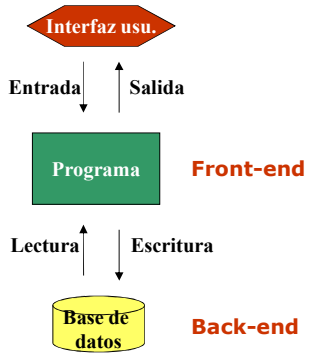
El modelo de programación ADO (y 2)

- 6. Opcionalmente, puede crear una vista del objeto de almacenamiento de forma que pueda navegar, ordenar y filtrar los datos (Recordset).
- 7. Edite los datos, ya sea añadiendo, eliminando o cambiando registros o datos (Recordset).
- 8. Si es apropiado, actualice el origen de datos con los cambios desde el objeto de almacenamiento (Recordset).
- 9. Si se utilizó una transacción, acepte o rechace los cambios hechos durante la transacción. Finalice la transacción (Connection).

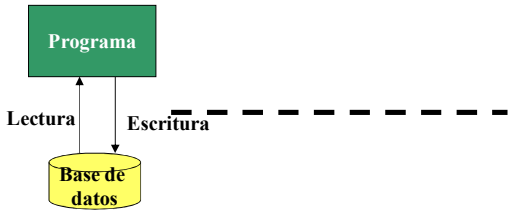
El modelo de programación ADO que usaremos

- 1. Haga una conexión a un origen de datos (Connection).
- 2. Ejecute un comando (Connection o Recordset).
- 3. Si el comando regresa registros, almacénelos en un objeto de almacenamiento (Recordset).
- 4. Edite los datos, ya sea añadiendo, eliminando o cambiando registros o campos (Recordset).
- 5. Si es apropiado, actualice el origen de datos con los cambios desde el Recordset.

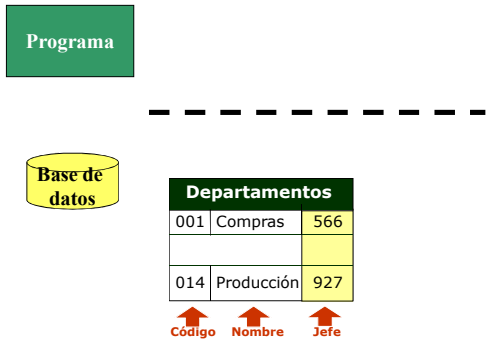
Ejemplo. Cambiar el nombre de departamento de compras



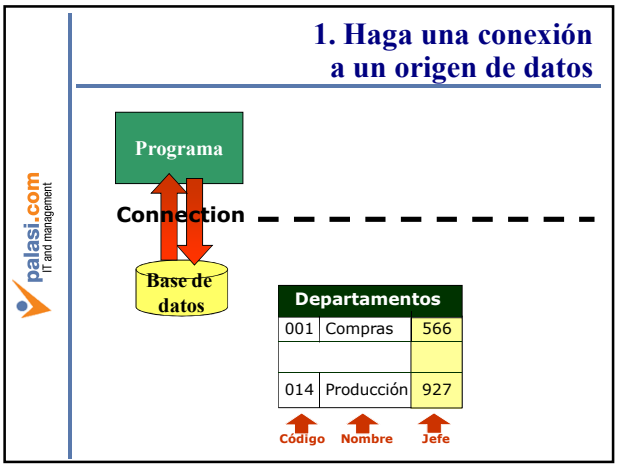
Ejemplo. Cambiar el nombre de departamento de compras

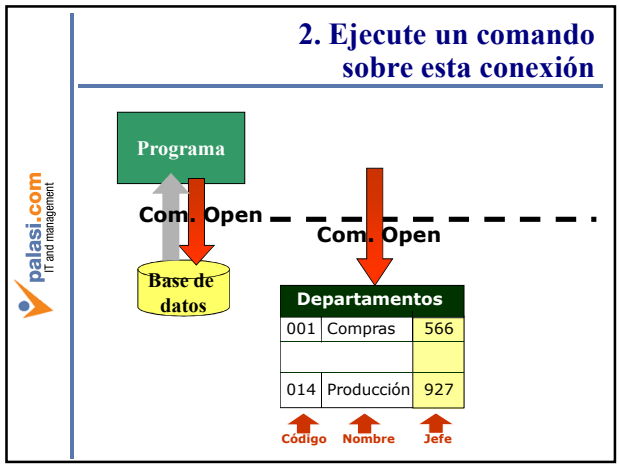


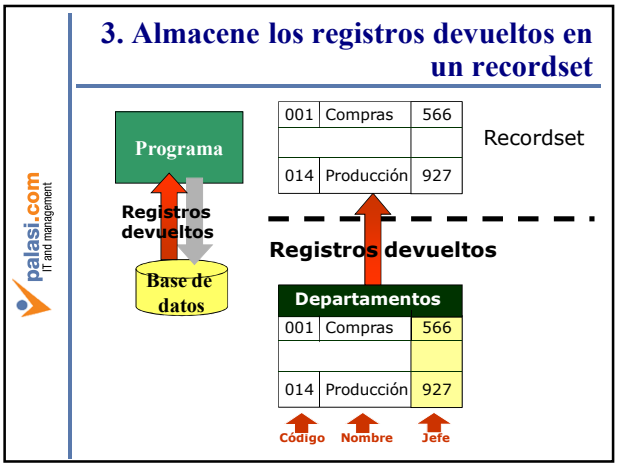
Ejemplo. Cambiar el nombre de departamento de compras

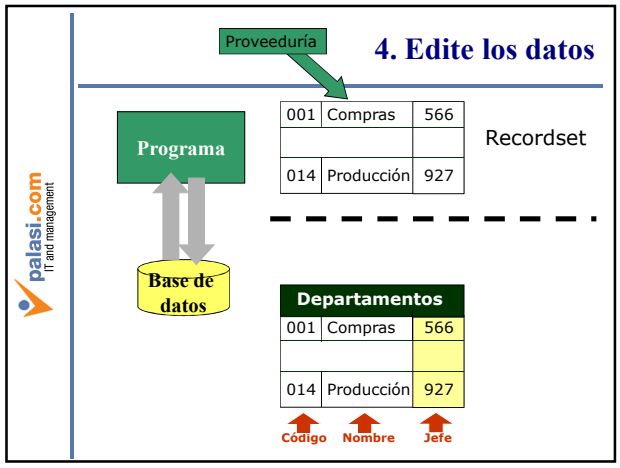


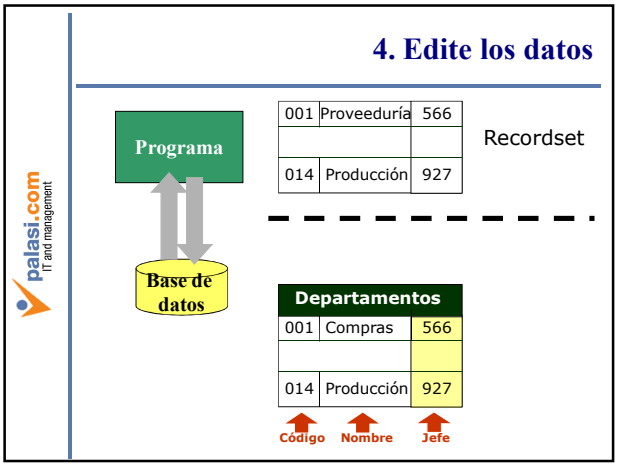
Mail: palasi@palasi.com Web: www.palasi.com

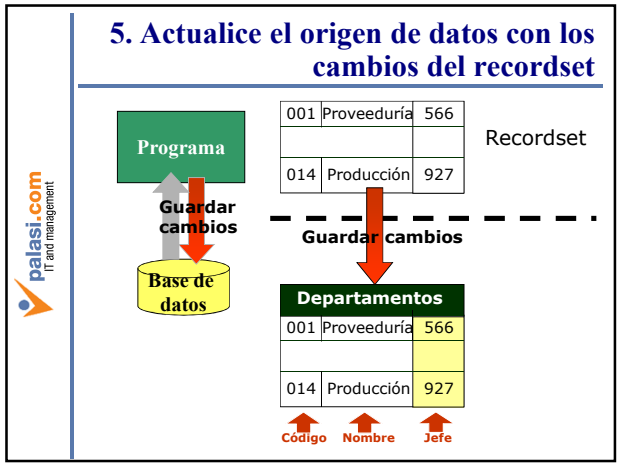












1. Haga una conexión a un origen de datos

- Comando:
  - *ObjetoConnection*.Open *CadenadeConnexión*

Abre la conexión a un origen de datos. La cadena de conexión especifica cuál es este origen de datos y las propiedades de conexión.

Ejemplo:

```
Dim cntConexionActual As New Connection
cntConexionActual.Open "Provider= _
Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD.mdb"
```

2. Ejecute comando y guarde los registros en recordset (1)

- Comando:
  - *ObjetoConnection*.Execute *ComandoSQL*

Ejecuta un comando SQL de modificación o borrado sobre la base de datos a la que se refiere la conexión.

Ejemplo (se supone que se ha definido la conexión cntConexionActual como antes).

```
cntConexionActual.Execute "Delete * From
Departamentos Where Nombre = 'Compras' "
```

2. Ejecute comando y guarde los registros en recordset (y 2)

```
ObjetoRecordset.CursorLocation = adUseClient
ObjetoRecordset.Open ComandoSQL, ConexionActiva, _
   adOpenStatic, adLockBatchOptimistic
Set ObjetoRecordset.ActiveConnection = Nothing
```

Ejecuta un comando SQL de consulta sobre la base de datos y se guardan los resultados en un recordset desconectado.

Ejemplo (se supone que se ha definido la conexión cntConexionActual como antes).

```
Dim rstRecordsetDepartamentos As ADODB.Recordset
rstRecordsetDepartamentos.CursorLocation = adUseClient
rstRecordsetDepartamentos.Open "Select * From
Departamentos", cntConexionActual, adOpenStatic,
adLockBatchOptimistic
Set ObjetoRecordset.ActiveConnection = Nothing
```

Sólo 1 línea

4. Edite los datos (1)

- Comando:
  - *ObjetoRecordset*.AddNew

Añade un registro en blanco.
Ejemplo: rstDepartamentos.AddNew

  - *ObjetoRecordset!NombreCampo = NuevoValor*

Fija el valor del campo especificado en el registro actual.
Ejemplo: rstDepartamentos!Nombre = "Proveeduría"

  - *Objeto*.Update

Confirma los cambios del registro actual en el recordset.
Ejemplo: rstDepartamentos.Update

  - *¿Qué pasa con los campos autonuméricos?*

4. Edite los datos (2)

- Procedimiento para modificar un registro

```
ObjetoRecordset!NombreCampo1 = NuevoValor1
...
ObjetoRecordset!NombreCampon = NuevoValorn
ObjetoRecordset.Update
```

```
ObjetoRecordset.AddNew
ObjetoRecordset!NombreCampo1 = NuevoValor1
...
ObjetoRecordset!NombreCampon = NuevoValorn
ObjetoRecordset.Update
```

¿Qué pasa con los campos autonuméricos al añadir?

4. Edite los datos (3)

- Comando:
  - *ObjetoRecordset*.CancelUpdate

Deshace los cambios del registro actual en el recordset.
Ejemplo: rstDepartamentos.CancelUpdate

  - *Objeto*.Delete

Borra el registro actual del recordset.
Ejemplo: rstDepartamentos.Delete

  - *Objeto*.MoveFirst

Hace que el registro actual sea el primero.
Ejemplo: rstDepartamentos.MoveFirst

Mail: palasi@palasi.com Web: www.palasi.com

4. Edite los datos (4)

- Comando:
  - *ObjetoRecordset*.MoveNext

Hace que el registro actual sea el siguiente al que es ahora.
Ejemplo: rstDepartamentos.MoveNext

  - *ObjetoRecordset*.Find *Criterio, 0, adSearchForward, adBookmarkFirst*

Hace que el registro actual sea el primero que cumple el criterio.
Ejemplo: rstDepartamentos.Find "Nombre = 'Compras' "

Indica si hemos llegado al final del recordet.
Ejemplo: rstDepartamentos.EOF

4. Edite los datos (y 5)

- Esquema de recorrido en ADO.

```
ObjetoRecordset.Movefirst
Do While Not ObjetoRecordset.EOF
    <Trata el registro actual de alguna manera>
    ObjetoRecordset.Movenext
Loop
```

5. Actualice el origen de datos con los cambios del recordset

ObjetoConnection.Open *CadenadeConnexión*
ObjetoRecordset.ActiveConnection = *ObjetoConnection*
ObjetoRecordset.MarshalOptions = adMarshalModifiedOnly
ObjetoRecordset.UpdateBatch
Set *ObjetoRecordset*.ActiveConnection = Nothing

Para actualizar la BD, abre una conexión, reconecta el recordset a la conexión y actualiza datos. Libera la conexión.
Ejemplo

```
Dim cntConexion As New Connection
cntConexion.Open "Provider= _
Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD.mdb"
rstRecordsetDepartamentos.ActiveConnection = cntConexion
rstRecordsetDepartamentos. MarshalOptions =
adMarshalModifiedOnly
rstRecordsetDepartamentos.UpdateBatch
Set ObjetoRecordset.ActiveConnection = Nothing
```

El modelo de programación ADO que usaremos

- 1. Haga una conexión a un origen de datos (Connection).
- 2. Ejecute un comando (Connection o Recordset).
- 3. Si el comando regresa registros, almacénelos en un objeto de almacenamiento (Recordset).
- 4. Edite los datos, ya sea añadiendo, eliminando o cambiando registros o campos (Recordset).
- 5. Si es apropiado, actualice el origen de datos con los cambios desde el Recordset.

Tareas comunes con recordsets desconectados

- 1. Recuperar el contenido de una tabla
- 2. Actualizar los cambios a una tabla.
- 3. Añadir un registro nuevo.
- 4. Modificar el contenido de un registro
- 5. Borrar un registro

1. Recuperar el contenido de una tabla

```
Function RecuperaTabla(strNomTabla As String) As
ADODB.Recordset
    Dim cntConexionActual As New Connection
    Dim rstRecordset As New ADODB.Recordset
    'Abre la conexión
    cntConexionActual.Open "Provider= _
    Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD.mdb"
    'Recupera un recordset desconectado con la tabla
    rstRecordset.CursorLocation = adUseClient
    rstRecordset.Open "Select * From " & strNomTabla, _
      cntConexionActual, adOpenStatic, adLockBatchOptimistic
    Set rstRecordset.ActiveConnection = Nothing
    'Retorna el recordset  obtenido
    Set Recu
End Function
```

Implementa los pasos 1, 2 y 3 del modelo

Mail: palasi@palasi.com Web: www.palasi.com

2. Actualizar los cambios a una tabla

```
Sub ActualizaTabla(rstRecordset As ADODB.Recordset)
    Dim cntConexionActual As New Connection
    'Abre una nueva conexión
    cntConexionActual.Open "Provider= _
    Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD.mdb"
    'Graba el recordset desconectado a la BD
    rstRecordset.ActiveConnection = cntConexionActual
    rstRecordset.MarshalOptions = _
      adMarshalModifiedOnly
    rstRecordset.UpdateBatch
    Set rstRecordset.ActiveConnection = Nothing
End Sub
```

Implementa el paso 5 del modelo

3. Añadir un registro nuevo

```
Dim rstRecordsetDepartamentos As ADODB.Recordset
'1,2y3.Recupera el contenido de la tabla departamentos
Set rstRecordsetDepartamentos = _
 RecuperaTabla("Departamentos")
'4.Añade un registro nuevo en blanco
rstRecordsetDepartamentos.AddNew
'4.Llena los campos (código se supone autonum.)
rstRecordsetDepartamentos!Nombre = "Publicidad"
rstRecordsetDepartamentos!Jefe = 678
'4.Fija los cambios en el recordset
rstRecordsetDepartamentos.Update
'5.Actualiza los cambios a la tabla
ActualizaTabla(rstRecordsetDepartamentos)
```

4. Modificar el contenido de un registro

```
Dim rstRecordsetDepartamentos As ADODB.Recordset
'1,2y3.Recupera el contenido de la tabla departamentos
Set rstRecordsetDepartamentos = _
 RecuperaTabla("Departamentos")
'4.Busca el registro que queremos modificar
rstRecordsetDepartamentos.Find "Nombre = 'Compras'", _
 0, adSearchForward, adBookmarkFirst
'4.Modifica el contenido de un campo
rstRecordsetDepartamentos!Nombre = "Proveeduría"
'4.Fija los cambios en el recordset
rstRecordsetDepartamentos.Update
'5.Actualiza los cambios a la tabla
ActualizaTabla(rstRecordsetDepartamentos)
```

5. Borrar un registro

```
Dim rstRecordsetDepartamentos As ADODB.Recordset
'1,2y3.Recupera el contenido de la tabla departamentos
Set rstRecordsetDepartamentos = _
 RecuperaTabla("Departamentos")
'4.Busca el registro que queremos borrar
rstRecordsetDepartamentos.Find "Nombre = 'Compras'", _
 0, adSearchForward, adBookmarkFirst
'4.Borra el registro del recordset
rstRecordsetDepartamentos.Delete
'5.Actualiza los cambios a la tabla
ActualizaTabla(rstRecordsetDepartamentos)
```

Modelo de programación ADO que utilizaremos (simplificado)

- 1. Recuperamos los registros de una tabla en un recordset desconectado.
- 2. Realizamos cambios sobre los datos en ese recordset.
- 3. Actualizamos los cambios del recordset sobre la taula.

Preguntas sobre ADO

Mail: palasi@palasi.com Web: www.palasi.com

Ejercicios

- Implementar código que realice las siguientes acciones:
  - Añada un cliente nuevo a la tabla clientes (inventar datos).
  - Modifique el cliente de código 22, para que su registro fiscal sea "112-345"
  - Borre el cliente de código 45.
- Pueden utilizar las funciones RecuperaTabla y ActualizaTabla.

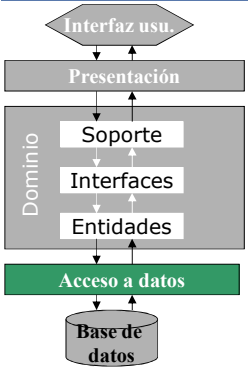
Laboratorio

- Implementar este código en el laboratorio.

Programa del seminario (1)

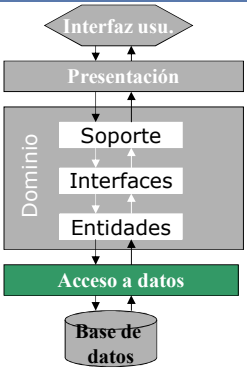
- 1. Objetivos y metodología.
- 2. Introducción a la programación n-capas.
- 3. Descripción del modelo que se usará.
- 4. Diseño de la base de datos.
- 5. Active Data Objects. Recordsets desconectados.
- 6. Capa de datos.
- 7. Capa de dominio: Entidades.
- 8. Capa de dominio: Interfaces de VB.

Capa de acceso a datos (1)



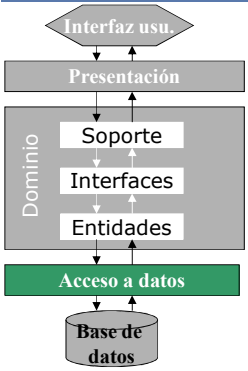
- Es la capa que accede directamente a la base de datos para recuperar y modificar datos de ella.
- Se suele colocar en la misma máquina que la base de datos o en máquinas cercanas, por rendimiento.

Capa de acceso a datos (2)



- Implementa un nivel de abstracción, de forma que, si cambiamos la base de datos o el método de acceso a ella, sólo tenemos que programar una nueva capa de datos y el programa sigue funcionando.

Capa de acceso a datos (y 3)



- En este caso, programaremos una capa de datos con una base de datos y la interfaz será ADO.

Mail: palasi@palasi.com Web: www.palasi.com

Capa de datos que implementaremos

- Estará basada en una sola clase, que llamaremos clsBD.
- Esta clase sólo tendrá dos métodos: uno permite recuperar un recordset de la BD **dada una sentencia SQL** y otro permite actualizar el recordset en la base de datos.

Indicaciones para la implementación en el laboratorio

- Para utilizar ADO en un proyecto de VB, hay que activar "Microsoft Active Data Objects 2.1 Library" en Proyecto-Referencias.
- Cada subcapa la programaremos en un proyecto diferente y la compilaremos como un EXE ActiveX (excepto la de presentación que será EXE estándar).
- Fijaremos la compatibilidad binaria del

Ejercicio

- Diseñar la capa de datos según esas indicaciones

Capa de datos. En archivo clsBD.cls

```
Option Explicit
Public Function RecuperaRst(strSQL As String) As
   ADODB.Recordset
      Dim cntConexionActual As New Connection
      Dim rstRecordset As New ADODB.Recordset
      'Abre la conexión
      cntConexionActual.Open "Provider= _
      Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD.mdb"
      'Recupera un recordset desconectado con la tabla
      rstRecordset.CursorLocation = adUseClient
      rstRecordset.Open strSQL, cntConexionActual, _
     adOpenStatic,adLockBatchOptimistic
      Set rstRecordset.ActiveConnection = Nothing
      'Retorna el recordset  obtenido
      Set RecuperaRst = rstRecordset
```

Capa de datos. En archivo clsBD.cls

```
(viene de página anterior)
Public Sub ActualizaRst(rstRecordset As ADODB.Recordset)
      Dim cntConexionActual As New Connection
      'Abre una nueva conexión
      cntConexionActual.Open "Provider= _
      Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD.mdb"
      'Graba el recordset desconectado a la BD
      rstRecordset.ActiveConnection = cntConexionActual
      rstRecordset.MarshalOptions = _
        adMarshalModifiedOnly
      rstRecordset.UpdateBatch
      Set rstRecordset.ActiveConnection = Nothing
End Sub
```

Cómo usar la capa de datos (1)

- Un esquema de uso podría ser el siguiente:

```
Dim objBD As Object
Dim rstRecordset As ADODB.Recordset
Set objBD = CreateObject("clsBD")
Set rstRecordset = objBD.RecuperaRst _
  (SentenciaSQLqueRecuperaElRecordset)

<Aquí realiza los cambios que se deseen en el
  recordset>

objBD.ActualizaRst(rstRecordset)
```

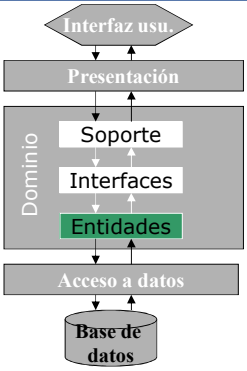
Cómo usar la capa de datos (y 2)

- El tipo Object se debe a que la definición de la clase pueda estar en otra máquina.
- En ese caso, en el CreateObject debemos indicar un identificador de la máquina.

Programa del seminario (1)

- 1. Objetivos y metodología.
- 2. Introducción a la programación n-capas.
- 3. Descripción del modelo que se usará.
- 4. Diseño de la base de datos.
- 5. Active Data Objects. Recordsets desconectados.
- 6. Capa de datos.
- 7. Capa de dominio: Entidades.
- 8. Capa de dominio: Interfaces de VB.

Subcapa de entidades



- Es la capa de dominio que está más en contacto con los datos.
- Su objetivo es modelizar las tablas que vamos a utilizar en forma de clases que podamos utilizar en las capas superiores.
- Se suele colocar en la misma máquina que la base de datos y la capa de acceso a datos, por

Entidad

- Clase que normalmente sirve para modelizar una tabla (normalmente, se sigue la correspondencia 1 entidad <-> 1 tabla)

Ejemplo de los asistentes al congreso

Asistentes

- CodAsist. A(6).
- Nombre. T(50).
- Apellidos T(50).
- Cedula N(20).
- CodPais A(3)
- CodTarifa A(3)
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

TiposTarifa

- CodTarifa A(3)
- TipoTarifa T(10)
- Importe N(3)

Paises

- CodPais A(3)
- Pais T(50)
- Avion L(1)

Cada tabla tendrá una entidad

- La tabla asistentes será modelizada por la entidad EntAsistentes.
- La tabla Paises por la entidad EntPaises.
- La tabla TiposTarifa por la entidad EntTiposTarifa.

Métodos de una entidad

- Sus métodos son las operaciones básicas que pueden realizarse sobre esta tabla:
  - Recuperar los registros de la tabla en un recordset.
  - Actualizar la tabla con los cambios realizados al recordset.
- Podrían haber más operaciones (depende del diseño) pero por ahora nos quedamos con éstas

Modelo de programación ADO que utilizaremos (simplificado)

- 1. Recuperamos los registros de una tabla en un recordset desconectado.
- 2. Realizamos cambios sobre los datos en ese recordset.

En capas superiores a la de entidades

- 3. Actualizamos los cambios del recordset sobre la taula.

Métodos de entidad Asistentes. Clase EntAsistentes.cls

```
Option Explicit
Public Function RecuperaAsistentes() As
    ADODB.Recordset
End Function

Public Sub ActualizaAsistentes(rstRecordset As
    ADODB.Recordset)
End Sub
```

Métodos de entidad Paises. Clase EntPaises.cls

```
Option Explicit
Public Function RecuperaPaises() As
    ADODB.Recordset
End Function

Public Sub ActualizaPaises(rstRecordset As
    ADODB.Recordset)
End Sub
```

Métodos de entidad TiposTarifa. Clase EntTiposTarifa.cls

```
Option Explicit
Public Function RecuperaTiposTarifa() As
    ADODB.Recordset
End Function

Public Sub ActualizaTiposTarifa(rstRecordset As
    ADODB.Recordset)
End Sub
```

¿Cómo se implementan esos métodos?

- No de la forma obvia y directa. P.ej., Accediendo a la base de datos mediante ADO y recuperando el recordset con un recordset.Open
- ¡¡Estamos en un modelo de programación en capas!!
- Cada capa se implementa usando los métodos de la anterior.
- En este caso, se implementarán usando los métodos de la capa de datos.

Métodos de la capa de datos

```
Option Explicit
Public Function RecuperaRst(strSQL As String) As
    ADODB.Recordset
        Dim cntConexionActual As New Connection
        Dim rstRecordset As New ADODB.Recordset
        'Abre la conexión
        cntConexionActual.Open "Provider= _
        Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD.mdb"
        'Recupera un recordset desconectado con la tabla
        rstRecordset.CursorLocation = adUseClient
        rstRecordset.Open strSQL, cntConexionActual, _
       adOpenStatic,adLockBatchOptimistic
        Set rstRecordset.ActiveConnection = Nothing
        'Retorna el recordset  obtenido
        Set RecuperaRst = rstRecordset
```

Métodos de la capa de datos

(viene de página anterior)

```
Public Sub ActualizaRst(rstRecordset As ADODB.Recordset)
        Dim cntConexionActual As New Connection
        'Abre una nueva conexión
        cntConexionActual.Open "Provider= _
        Microsoft.Jet.OLEDB.4.0;Data Source=C:\BD.mdb"
        'Graba el recordset desconectado a la BD
        rstRecordset.ActiveConnection = cntConexionActual
        rstRecordset.MarshalOptions = _
          adMarshalModifiedOnly
        rstRecordset.UpdateBatch
    End Sub
```

Nos olvidamos de la implementación

- En la programación orientada a objetos, no nos interesan cómo están implementados los métodos a la hora de usarlos (recordar la "Ocultación de la implementación").
- En la programación en capas no nos interesa como está implementada una capa cuando la usamos desde una capa superior

Métodos de la capa de datos (sin implementación)

```
Option Explicit
Public Function RecuperaRst(strSQL As String) As
    ADODB.Recordset
End Function

Public Sub ActualizaRst(rstRecordset As
    ADODB.Recordset)
End Sub
```

Implementación de los métodos de EntAsistentes

- Como hemos dicho, los implementaremos a partir de los métodos de la capa de datos.
- El RecuperaAsistentes lo implementaremos a partir de RecuperaRst.
- El ActualizaAsistentes a partir del ActualizaRst.

Implementación de RecuperaAsistentes (1)

```
Public Function RecuperaAsistentes() As
    ADODB.Recordset
    ...
    objBD.RecuperaRst(...)
    ...
End Function
```

- los métodos deben invocarse de forma <objeto>.<método>. ObjBD deberá ser de la clase clsBD

Mail: palasi@palasi.com Web: www.palasi.com

Implementación de RecuperaAsistentes (2)

```
Public Function RecuperaAsistentes() As
    ADODB.Recordset
    Dim objBD As Object
    Set objBD = CreateObject("clsBD")
    objBD.RecuperaRst(...)
    Set objBD = Nothing
    ...
End Function
```

- Se crea 1 obj. de clase clsBD para usar RecuperaRst.El tipo es Object para permitir capa datos en otra máquina

Implementación de RecuperaAsistentes (3)

```
Public Function RecuperaAsistentes() As
    ADODB.Recordset
    Dim objBD As Object
    Set objBD = CreateObject("clsBD")
    objBD.RecuperaRst("Select * From Asistentes")
    Set objBD = Nothing
    ...
End Function
```

- El parámetro es la sentencia SQL que recupera el contenido de la tabla

Implementación de RecuperaAsistentes (4)

```
Public Function RecuperaAsistentes() As
    ADODB.Recordset
    Dim objBD As Object
    Set objBD = CreateObject("clsBD")
    Set RecuperaAsistentes =
    objBD.RecuperaRst("Select * From Asistentes")
    Set objBD = Nothing
End Function
```

- El resultado de RecuperaAsistentes debe ser el resultado de RecuperaRst

Implementación de Actualiza Asistentes

```
Public Sub ActualizaAsistentes(rstRecordset As
    ADODB.Recordset)
    Dim objBD As Object
    Set objBD = CreateObject("clsBD")
    objBD.ActualizaRst(rstRecordset)
    Set objBD = Nothing
End Sub
```

- Se utiliza ActualizaRst. Se utiliza el mismo esquema que antes.

Preguntas sobre entidades

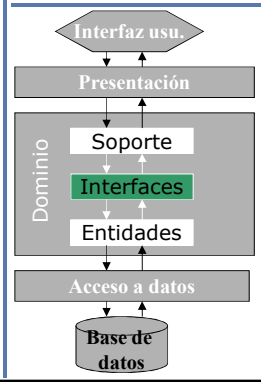
Ejercicio

- Implementar las entidades correspondientes a las tablas Facturas, Clientes y TipoDocumento del ejercicio anterior.

Programa del seminario (1)

- 1. Objetivos y metodología.
- 2. Introducción a la programación n-capas.
- 3. Descripción del modelo que se usará.
- 4. Diseño de la base de datos.
- 5. Active Data Objects. Recordsets desconectados.
- 6. Capa de datos.
- 7. Capa de dominio: Entidades.
- 8. Capa de dominio: Interfaces de VB.

Subcapa de interfaces



- Es la subcapa de dominio inmediatamente superior a la de entidades.
- Su objetivo es dar una misma forma para operar con todas las entidades
- Es la única forma de conseguir polimorfismo en Visual Basic, ya que no hay herencia.

Polimorfismo

- Es la capacidad de acceder a varias clases con los mismos métodos y propiedades pero de obtener un comportamiento diferente según cada clase.
- Ejemplo: Tres clases llamadas Rectángulo, Círculo y Triángulo con un método Dibujar, de forma que objCírculo.Dibujar dibujará un círculo y objTriángulo.Dibujar dibujará un triángulo.

La importancia del polimorfismo

- Permite acceder a clases diferentes de la misma manera.
- Es útil cuando necesitamos un compromiso entre clases diferentes pero similares.
- Por ejemplo, cuando tenemos un objeto que no sabemos de qué clase es pero sí las características de la clase. Ej: objFiguraGeométrica.
- Por ejemplo, tareas de mantenimiento.

Cómo implementar polimorfismo

- Los lenguajes orientados a objetos implementan el polimorfismo con un mecanismo denominado "herencia de implementación" o simplemente "herencia"
- Pero Visual Basic (hasta la versión 6) no soporta "herencia de implementación" sino la llamada "herencia de interfaz".

Herencia de interfaz

- Llamaremos interfaz a un conjunto de propiedades y métodos con los cuales se puede acceder a una clase.
- Todas las clases en Visual Basic tienen como mínimo una interfaz pero pueden tener más.

Todas las clases tienen como mínimo 1 interfaz

- La clase Círculo tiene como interfaz las funciones Centro y Circunferencia.
- Pero cómo hacemos que haya más de una interfaz.

Clase Círculo

```
Option Explicit
Public Function Centro As Punto
...
End Function
Public Function Circunferencia As Double
...
End Function
```

Cómo hacer para tener más de 1 interfaz (1)

- Definimos una nueva clase que no tiene implementación (clase abstracta) con los métodos y propiedades correspondiente a la interfaz que queremos definir.
- Los métodos son propios de cualquier figura.

Clase Figura

```
Option Explicit
Public Sub Dibujar
End Sub
Public Function Area As Double
End Function
```

Cómo hacer para tener más de 1 interfaz (2)

- Indicamos que la clase Círculo "hereda" la interfaz definida en la clase Figura.

Clase Círculo

```
Option Explicit
Implements Figura
Public Function Centro As Punto
End Function
Public Function Circunferencia As Double
End Function
```

Cómo hacer para tener más de 1 interfaz (3)

Clase Círculo

```
Option Explicit
Implements Figura
Public Function Centro As Punto
Public Function Circunferencia ...
Private Sub Figura_Dibujar
<implementación de Dibujar para Círculo>
End Sub
Private Function Figura_Area As Double
<implementación de Área para Círculo>
End Function
```

- Implementamos los métodos de la interfaz definida en la clase abstracta (como Private) .

Cómo hacer para tener más de 1 interfaz (4)

Clase Círculo

```
Option Explicit
Implements Figura
Public Function Centro As Punto
Public Function Circunferencia ...
Private Sub Figura_Dibujar
<implementación de Dibujar para Círculo>
End Sub
Private Function Figura_Area As Double
<implementación de Área para Círculo>
End Function
```

- Fíjense que el nombre de los métodos contiene el nombre de la clase que define la interfaz .

Cómo hacer para tener más de 1 interfaz (y 5)

- Ahora tenemos dos interfaces: la original de la clase Círculo y la que hemos heredado e implementado de la clase Figura.

Clase Círculo

```
Option Explicit
Implements Figura
Public Function Centro As Punto
Public Function Circunferencia As ...
Private Sub Figura_Dibujar
Private Function Figura_Area As Double
```


Mail: palasi@palasi.com Web: www.palasi.com

Cómo utilizar varias interfaces (1)

- Supongamos un objeto de clase Circulo.

```
Dim objCirculo As Circulo
```

Clase Círculo

```
Option Explicit
Implements Figura
Public Function Centro As Punto
Public Function Circunferencia As ...
Private Sub Figura_Dibujar
Private Function Figura_Area As Double
```

Cómo utilizar varias interfaces (2)

- Para utilizar los métodos de la interfaz predeterminada de Círculo.

```
objCirculo.Centro
objCirculo.Circunferencia
```

Clase Círculo

```
Option Explicit
Implements Figura
Public Function Centro As Punto
Public Function Circunferencia As ...
Private Sub Figura_Dibujar
Private Function Figura_Area As Double
```

Cómo utilizar varias interfaces (3)

- Para utilizar los métodos de la interfaz heredada de Figura.

```
Dim IFigura As Figura
Set IFigura = objCirculo
IFigura.Dibujar
IFigura.Area
```

Clase Círculo

```
Option Explicit
Implements Figura
Public Function Centro As Punto
Public Function Circunferencia As ...
Private Sub Figura_Dibujar
Private Function Figura_Area As Double
```

Cómo utilizar varias interfaces (y 4)

- Para utilizar los métodos de la interfaz heredada de Figura.

```
Dim IFigura As Figura
Set IFigura = objCirculo
IFigura.Dibujar
IFigura.Area
```

- Fíjense que para utilizar la interfaz debemos de definir un objeto de la clase predeterminada (no abstracta) Circulo y asignarlo a una variable de la clase

Observaciones sobre las interfaces

- Aunque el método se llama igual para todas las clases, se implementa de forma específica
- Podemos acceder a clases similares pero diferentes de la misma manera.
- También sirve para realizar diversas versiones de las clases.
- Implementa una especie de herencia.

¿Para qué necesitamos las interfaces en nuestro caso?

- Todas las entidades de la subcapa de entidades modelizan tablas.
- Nos interesaría tener unos métodos de acceso generales para tablas con el fin de acceder de la misma manera a las diferentes entidades.
- Esto puede ser útil en procesos que implican diversas entidades (ejemplo, purgado de información antigua).

Mail: palasi@palasi.com Web: www.palasi.com

Clase abstracta que definiremos

- Recordemos que es una clase abstracta: no tiene implementación, sólo sirve para definir la interfaz

Clase ITabla

```
Option Explicit
Public Function Recupera () As ADODB.Recordset

End Function

Public Sub Actualiza(rstRecordset As
ADODB.Recordset)
End Sub
```

Heredar la interfaz

- Se hereda la interfaz definida en la clase ITabla.

Clase EntAsistentes

```
Option Explicit
Implements ITabla
Private Function RecuperaAsistentes() As
ADODB.Recordset
Private Sub ActualizaAsistentes(rstRecordset As
ADODB.Recordset)
Private Function ITabla_Recupera
(strNomTabla As String) As ADODB.Recordset

Private Sub ITabla_Actualiza(rstRecordset As
ADODB.Recordset)
```

Implementar los métodos de la interfaz

```
Option Explicit
Implements ITabla
Private Function RecuperaAsistentes() As
ADODB.Recordset
Private Sub ActualizaAsistentes(rstRecordset As
ADODB.Recordset)
Private Function ITabla_Recupera () As
ADODB.Recordset
    Set ITabla_Recupera = RecuperaAsistentes
End Function
Private Sub ITabla_Actualiza(rstRecordset As
ADODB.Recordset)
    ActualizaAsistentes rstRecordset
End Sub
```

Cómo llamar a la interfaz ITabla

- Supongamos que queremos llamar a la interfaz ITabla desde un objeto de entidad Asistentes.
- Dim objAsistentes As EntAsistentes
- Dim ITablaAct As Itabla
- Set ITablaAct = objAsistentes
- ITablaAct.Actualiza(rstRecordsetActual)

Implementar la interfaz ITabla

- Con las otras entidades es lo mismo.
- 1. Se implementa la clase abstracta ITabla
- 2. En cada entidad se coloca Implements ITabla
- 3. En cada entidad se implementan los métodos de la interfaz llamando a los métodos de la entidad.

Ejercicio

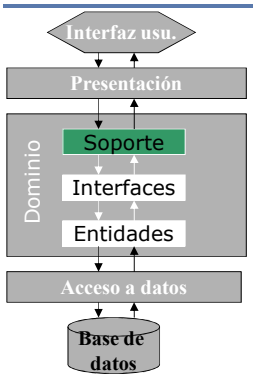
- Implementar la interfaz Itabla para las entidades EntFacturas, EntClientes y EntTipoDocumento

Mail: palasi@palasi.com Web: www.palasi.com

Programa del seminario (y 2)

- 9. Capa de dominio: Clases de Soporte.
- 10. Capa de presentación.
- 11. Otros aspectos.
- 12. Conclusión.

Subcapa de soporte



- Es la subcapa de dominio más cercana a la capa de presentación.
- Su objetivo es proveer los métodos para que los formularios de la capa de presentación puedan acceder a los datos
- Cada clase de soporte se corresponde con un formulario de la capa de presentación

Cada clase de soporte se corresponde a un formulario

- Contiene métodos de recuperación y actualización de todas las tablas que intervienen en un formulario.
- Accede a todas esas tablas a través de las entidades accedidas mediante una interfaz (en nuestro caso, la interfaz ITabla)

Ejemplo de formulario (1)

- Formulario de inscripción de asistentes.
- Objetivo: Introducir y consultar la información correspondiente a los asistentes de un congreso.
- Funciones: Poder consultar los datos de un asistente, navegar entre los diversos asistentes, introducir un asistente nuevo, modificar o eliminar un asistente existente.

Ejemplo de formulario (2)

- Datos que intervienen en el formulario: nombre, apellidos, número de cédula, nacionalidad, tipo de tarifa, teléfono, dirección y hotel.
- Tablas que intervienen en el formulario:
  - Tabla Asistentes: contiene los datos mencionados.
  - Tabla Paises y TiposTarifa. Sirven para llenar los cuadros desplegables de los campos respectivos.

Ejemplo de formulario (y 3)

Tablas que intervienen en el formulario de ejemplo

- Tabla "Asistentes". Para rellenar los campos del formulario.
- Tablas "Paises" y "TiposTarifa". Para rellenar las listas desplegables.
- Son las mismas que deben aparecer en la clase de soporte correspondiente al formulario.

Tablas que aparecen en la clase de soporte

- "Asistentes", "Paises" y "TiposTarifa".
- Las operaciones que debemos hacer con cada una de estas tablas son las de siempre: Recuperar y Actualizar.
- La clase de soporte debe tener seis métodos:
  - RecuperaAsistentes, ActualizaAsistentes
  - RecuperaPaises.RecuperaTiposTarifa.

  (no actualizamos las tablas auxiliares)

Esquema de la clase de soporte SopAsistentes

```
Option Explicit
Public Function RecuperaAsistentes() As
ADODB.Recordset

Public Sub ActualizaAsistentes(rstRecordset As
ADODB.Recordset)

Public Function RecuperaPaises() As
ADODB.Recordset

Public Function RecuperaTiposTarifa() As
ADODB.Recordset
```

¿Cómo implementamos estos métodos?

- Como estamos en un diseño de tres capas, los métodos deberán ser implementados a partir de la capa inferior.
- En este caso, deberemos implementarlos a partir de la capa de interfaces.

Implementación de Recupera Asistentes

```
Option Explicit
Public Function RecuperaAsistentes() As
   ADODB.Recordset

   Dim objAsistentes As EntAsistentes
   Dim ITablaActual As Itabla

   'Crea un objeto de la entidad Asistentes y lo
   'asigna a variable interfaz COMO HEMOS VISTO
   Set objAsistentes = New EntAsistente
   Set ITablaActual = objAsistentes

   'Se llama al método Recupera de la Interfaz
   Set RecuperaAsistentes = ITablaActual.Recupera()

End Function
```

Implementación de ActualizaAsistentes

```
Option Explicit
Public Sub ActualizaAsistentes(rstRecordset As
   ADODB.Recordset)
   Dim objAsistentes As EntAsistentes
   Dim ITablaActual As Itabla

   'Crea un objeto de la entidad Asistentes y lo
   'asigna a variable interfaz COMO HEMOS VISTO
   Set objAsistentes = New EntAsistente
   Set ITablaActual = objAsistentes

   'Se llama al método Actualiza de la Interfaz
   ITablaActual.Actualiza rstRecordset
```


Mail: palasi@palasi.com Web: www.palasi.com

¿Cómo implementar los otros métodos de SopAsistentes?

- De la misma manera que los métodos que acabamos de describir.
- La única diferencia es que lo haremos respecto a objetos de las entidades EntPaises y EntTiposTarifa en vez del objeto objAsistentes de la entidad EntAsistentes.

Un pequeño misterio

- Con lo que acabamos de ver hemos acabado la capa de dominio.
- Habíamos afirmado que la capa de dominio se dedicaba al cálculo de la aplicación.
- Sin embargo, no hemos visto que se haga ningún cálculo en esta capa. Sólo se propagan los datos a la capa de presentación. ¿Qué pasa?

Solución del misterio

- La tarea que estamos programando (un formulario para la introducción de datos) prácticamente no tiene cálculo.
- Es por ello que el cálculo no aparece en la capa de dominio.

Tareas que comprenden cálculo

- Tareas que sí comprenden cálculo:
  - Generación de reportes (deben calcularse los datos del reporte a partir de las tablas de la BD).
  - Procesos de cálculo sobre las tablas (por ejemplo, cierre del año contable, actualización de partidas, etc).
- En este caso, el cálculo se programa en la subcapa de soporte de la capa de dominio.

Preguntas sobre la subcapa de soporte

Ejercicio

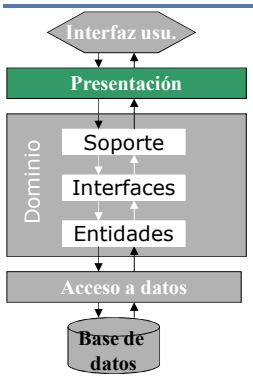
- Implementar la capa de soporte para el un formulario de facturas. Las tablas y campos involucrados son los siguientes:
  - TablaAsistentes (para rellenar los campos del formulario que son: tipo de documento, fecha, número de documento, el cliente, sucursal, si es al crédito, el total de ventas exentas, el total de ventas gravadas)
  - TablaClientes. Para seleccionar los clientes de una lista desplegable.

Mail: palasi@palasi.com Web: www.palasi.com

Programa del seminario (y 2)

- 9. Capa de dominio: Clases de Soporte.
- 10. Capa de presentación.
- 11. Otros aspectos.
- 12. Conclusión.

Capa de presentación



- Es la capa que genera la interfaz de usuario (p. ej, los formularios con los cuales el programa se comunica con el usuario)
- Recupera los datos de capas inferiores y los presenta en un formato amigable al usuario.
- Captura las entradas del usuario y la pasa a las

Las funciones de la capa de presentación

- Usando las capas inferiores:
  - 1. Recuperar registros de capas inferiores.
  - 2. Guardar registros a las capas inferiores.
- Sin acceder a ninguna otra capa:
  - 3. Mostrar los datos de un registro.
  - 4. Desplazarse entre los registros.
  - 5. Añadir un registro nuevo.
  - 6. Eliminar un registro existente.
  - 7. Modificar un registro existente.

Conclusión obvia

- La capa de presentación suele ser más compleja que las restantes capas.
- Hay que dedicar más tiempo en la programación para ella.

Todas estas operaciones se realizan mediante ADO

- Usando las capas inferiores:
  - 1. Recuperar registros de capas inferiores.
  - 2. Guardar registros a las capas inferiores.
- Sin acceder a ninguna otra capa:
  - 3. Mostrar los datos de un registro.
  - 4. Desplazarse entre los registros.
  - 5. Añadir un registro nuevo.
  - 6. Eliminar un registro existente.
  - 7. Modificar un registro existente.

Todas estas operaciones se realizan mediante ADO

- Se define un recordset ADO para cada formulario que contiene los registros que se desean consultar.
- Este recordset se recupera en el evento Form_Load y se manipula con los restantes eventos.
- Se define como variable de la clase con el estilo:

Private rstAsistentes As ADODB.Recordset

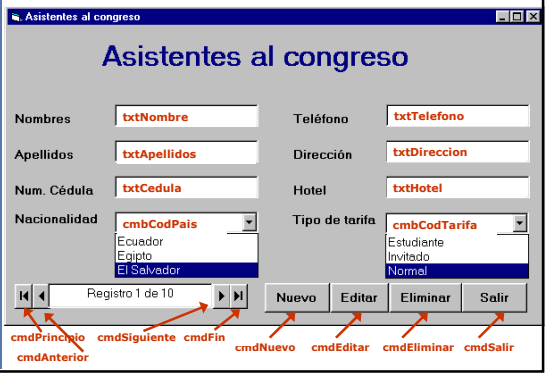
Mail: palasi@palasi.com Web: www.palasi.com

Implementación de estas propiedades

- Usando las capas inferiores:
  - 1. Recuperar registros de capas inferiores.
  - 2. Guardar registros a las capas inferiores.
- Sin acceder a ninguna otra capa:
  - 3. Mostrar los datos de un registro.
  - 4. Desplazarse entre los registros.
  - 5. Añadir un registro nuevo.
  - 6. Eliminar un registro existente.
  - 7. Modificar un registro existente.

Ejemplo de formulario

Ejemplo de formulario



1. Recuperar registros de capas inferiores

- Comando:
  - Se utiliza el método correspondiente la la clase de Soporte.
- Evento:
  - Form_Load

Ejemplo:

```
Dim objSopAsistentes As New SopAsistentes
Set rstAsistentes = _
  objSopAsistentes.RecuperaAsistentes()
```

2. Guardar registros a las capas inferiores

- Comando:
  - Se utiliza el método correspondiente la la clase de Soporte.
- Eventos:
  - (Lo veremos más adelante)

Ejemplo:

```
objSopAsistentes.ActualitzaAsistentes _
        (rstAsistentes)
```

3. Mostrar los datos existentes (1)

- Se utilizan las siguientes propiedades.
- Para TextBox y DataCombo.
  - DataSource: contiene una referencia al recordset al que está enlazado el control.
  - DataField: contiene el nombre del campo del recordset al que está enlazado el control.
- Sólo para DataCombo.
  - RowSource: contiene una referencia al recordset que llena la lista del DataCombo.
  - ListItem: contiene el nombre del campo que aparece en la lista.
  - BoundColumn: nombre de campo que suministra

3. Mostrar los datos existentes (2)

- Comando:
  - Se encadena el control que muestra el dato con el recordset que se ha recuperado mediante las propiedades DataSource y DataField.
- Evento:
  - Form_Load

Ejemplo:
Me.txtNombre.DataField = "Nombres"
Set Me.txtNombre.DataSource = rstAsistentes.

3. Mostrar los datos existentes (3)

- Debe rellenarse las listas de los cuadros combinados con RowSource, BoundColumn y ListField
- Evento:
  - Form_Load

Ejemplo:
Set Me.cmbCodtarifa.RowSource = rstTiposTarifas
Me. cmbCodtarifa.ListField = "TipoTarifa"
Me. cmbCodtarifa.BoundColumn = "CodTarifa"

4. Desplazarse entre los registros

- Comando:
  - Se utilizan las operaciones del recordset ADO (que ya vimos): MoveFirst, MoveLast, MoveNext, MovePrevious.
- Eventos:
  - cmdSiguiente_Click, cmdAnterior_Click
  - cmdPrincipi_Click, cmdFi_Click

Ejemplo:
(cmdSiguiente): Me.rstAsistentes.MoveNext
(cmdAnterior): Me.rstAsistentes.MovePrevious
(cmdPrincipi): Me.rstAsistentes.MoveFirst
(cmdFi): Me.rstAsistentes.MoveLast

5. Añadir un registro nuevo

- Comando:
  - Se utiliza la operación AddNew del recordset ADO (que ya vimos). También deben guardarse los cambios en la base de datos pero sólo después de que se modifiquen (lo dejamos para después).
- Eventos:
  - cmdNuevo_Click

Ejemplo:
Me.rstAsistentes.AddNew

Guardar los datos de un registro nuevo en la BD (1)

- Secuencia:
  - 1. Crear un registro en blanco.
  - 2. Dejar que el usuario introduzca los datos.
  - 3. Guardar los cambios (guardar en el recordset y guardar en la BD).
- Dos modalidades:
  - 1. Con un botón especial para guardar los cambios. Formulario modal
  - 2. Se guardan los cambios con cualquier botón que se presiona después de introducirlos. Formulario no modal.

Guardar los datos de un registro nuevo en la BD (2)

- ¿Qué tipo de formulario elegir?
  - El formulario modal es más sencillo, más eficiente y da más control al usuario.
  - Necesita ser programado con un poco más de código.
- En esta presentación elegiremos el formulario no modal por cuestiones de concisión.
- No necesita botón de "Editar".

Guardar los datos de un registro nuevo en la BD (y 3)

- Versión de Formulario No Modal. Cada botón tendrá en su evento Click.

```
If <DatosIncorrectos> Then
   <Emitir Mensaje de Error>
Else
   rstAsistentes.Update
   objSopAsistentes.ActualitzaAsistentes _
     (rstAsistentes)
EndIf
```

6. Eliminar un registro existente

- Comando:
  - Se utiliza la operación Delete del recordset ADO (que ya vimos). También deben guardarse los cambios en la base de datos (lo implementamos con la solución acabada para después).
- Eventos:
  - cmdEliminar_Click

Ejemplo:
Me.rstAsistentes.Delete

5. Editar un registro existente

- No se necesita un método ADO para editar un registro.
- Por ello, en un formulario no modal no se necesita un botón "Editar" para editar un registro: se puede hacer directamente.

Aplicando todos estos conocimientos

- Vamos a aplicar todos estos conocimientos mostrando el código completo del formulario de Asistentes.
- Por simplicidad, prescindiremos de la validación de los datos.

Apariencia del formulario Asistentes

Declaraciones del formulario

```
Option Explicit
'Recordset de Asistentes
Private rstAsistentes As ADODB.Recordset
'Declaración de la clase de soporte
Private objSopAsistentes As SupAsistentes
```

Form_Load (1)

```
Private Sub Form_Load()
'Inicializamos el objeto de soporte
Dim objSopAsistentes As New SopAsistentes

'Recuperamos los recordsets del formulario
Set rstAsistentes = _
objSopAsistentes.RecuperaAsistentes()
Set rstPaises = _
objSopAsistentes.RecuperaPaises()
Set rstTiposTarifa = _
objSopAsistentes.RecuperaTiposTarifa()
```

Form_Load (2)

```
'Enlazamos más controles
Me.txtNombre.DataField = "Nombres"
Set Me.txtNombre.DataSource = rstAsistentes
Me.txtApellidos.DataField = "Apellidos"
Set Me.txtNombre.DataSource = rstAsistentes
Me.txtCedula.DataField = "Cedula"
Set Me.txtCedula.DataSource = rstAsistentes
Me.cmbCodPais.DataField = "CodPais"
Set Me.cmbCodPais.DataSource = rstAsistentes
Me.txtTelefono.DataField = "Telefono"
Set Me.txtTelefono.DataSource = rstAsistentes
```

Form_Load (3)

```
'Enlazamos más controles
Me.txtDireccion.DataField = "Direccion"
Set Me.txtDireccion.DataSource = rstAsistentes
Me.txtHotel.DataField = "Hotel"
Set Me.txtHotel.DataSource = rstAsistentes

'Rellenamos las listas de los cuadros combinados
Set Me.cmbCodPais.RowSource = rstPaises
Me. cmbCodPais.ListField = "Pais"
Me. cmbCodPais.BoundColumn = "CodPais"
```

Form_Load (y 4)

```
'Rellenamos las listas de los cuadros combinados
Set Me.cmbCodTarifa.RowSource = rstTiposTarifa
Me.cmbCodTarifa.ListField = "TipoTarifa"
Me.cmbCodTarifa.BoundColumn = "CodTarifa"

End Sub
```

cmdPrincipio_Click

```
Private Sub cmdPrincipio_Click()
   'Guardamos los cambios
   rstAsistentes.Update
   objSopAsistentes.ActualitzaAsistentes _
     (rstAsistentes)

   'Nos colocamos al principio del recordset
   rstAsistentes.MoveFirst
End Sub
```

cmdAnterior_Click

```
Private Sub cmdAnterior_Click()
   'Guardamos los cambios
   rstAsistentes.Update
   objSopAsistentes.ActualitzaAsistentes _
     (rstAsistentes)

   'Nos colocamos en el registro anterior
   rstAsistentes.MovePrevious
End Sub
```

cmdSiguiente_Click

```
Private Sub cmdSiguiente_Click()
   'Guardamos los cambios
   rstAsistentes.Update
   objSopAsistentes.ActualitzaAsistentes _
     (rstAsistentes)

   'Nos colocamos en el registro siguiente
   rstAsistentes.MoveNext
End Sub
```

cmdFin_Click

```
Private Sub cmdFin_Click()
   'Guardamos los cambios
   rstAsistentes.Update
   objSopAsistentes.ActualitzaAsistentes _
     (rstAsistentes)

   'Nos colocamos al final del recordset
   rstAsistentes.MoveLast
End Sub
```

cmdNuevo_Click

```
Private Sub cmdNuevo_Click()
   'Guardamos los cambios
   rstAsistentes.Update
   objSopAsistentes.ActualitzaAsistentes _
     (rstAsistentes)

   'Creamos un registro en blanco
   rstAsistentes.AddNew
End Sub
```

cmdEliminar_Click

```
Private Sub cmdEliminar_Click()
   'Guardamos los cambios
   rstAsistentes.Update
   objSopAsistentes.ActualitzaAsistentes _
     (rstAsistentes)

   'Eliminamos el registro
   rstAsistentes.Delete
End Sub
```

cmdSalir_Click

```
Private Sub cmdSalir_Click()
   'Guardamos los cambios
   rstAsistentes.Update
   objSopAsistentes.ActualitzaAsistentes _
     (rstAsistentes)

   'Salimos del formulario
   Unload Me
End Sub
```

Ejercicio

- Realizar la capa de presentación del ejemplo de las facturas.

Preguntas sobre la capa de presentación

Programa del seminario (y 2)

- 9. Capa de dominio: Clases de Soporte.
- 10. Capa de presentación.
- 11. Otros aspectos.
- 12. Conclusión.

Lo que se ha implementado es un ejemplo "de juguete"

- No se podría haber realizado un ejemplo de tamaño industrial. Sólo una introducción para comenzar.
- Hay aspectos que se han dejado de lado.
- Se quieren destacar tres: control de errores, compilación y XML.

Control de errores

- En un programa tradicional de VB.

```
Public Sub XXX
    On Error Goto TrataError
    ...

    Exit Sub
TrataError:
    MsgBox("Error " & Str(Err.Number))
    Exit Sub
End Sub
```

En caso de error, emite un mensaje

Pero esto no es posible en una aplicación de 3 capas

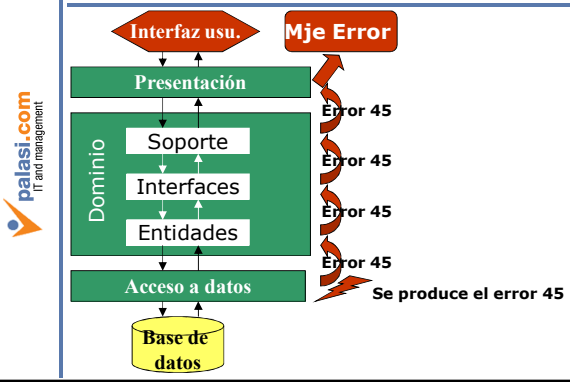
- Imaginemos que este código está en la capa de datos.
- Dentro de la capa de datos, accederíamos a la interfaz del usuario (MsgBox).
- ¡Esto violaría el principio de separación de las capas!

Solución: Propagación de errores

- En vez de emitir el mensaje de error, lo propagamos.
- De la capa de datos, el número de error pasa a la capa de dominio y de allí a la capa de presentación.
- Una vez se encuentra en la capa de presentación, podemos emitir el mensaje correspondiente.

Mail: palasi@palasi.com Web: www.palasi.com

Propagación de errores



¿Cómo se implementa esto? (1)

- 1. Cada clase tiene una propiedad llamada NumError que contiene el Numero de Error (0 si no hay error).
- 2. Cada vegada que se produce un error se cambia la propiedad NumError.

¿Cómo se implementa esto? (2)

- En un programa en 3 capas (clase clsCapaX)

```
Option Explicit
Public Sub MetodoX
    On Error Goto TrataError
    ...

    Exit Sub
TrataError:
    Me.NumError = Err.Number
    Exit Sub
End Sub
```

En caso de error, fija la propiedad NumError

¿Cómo se implementa esto? (3)

- 3. Cada vez que se llama a un método de una clase inferior se consulta la propiedad NumError de la clase inferior.
- 4. Se fija el valor de NumError de la clase actual como el que tiene la clase inferior.
- 5. Una vez en la capa de presentación, se emite el mensaje de error correspondiente a la propiedad NumError.

¿Cómo se implementa esto? (4)

- **La clase clsCapaY llama a la clase clsCapaX(de una capa inferior)**

```
Option Explicit
Public Sub MetodoY
    On Error Goto TrataError
    ...
    objclsX.MetodoX
    If obclsX.NumError <> 0 Then
       Err.Raise obclsX.NumError
    EndIf
    Exit Sub
TrataError:
    Me.NumError = Err.Number
```

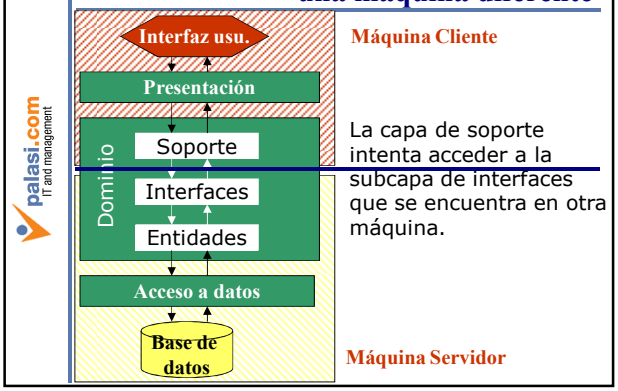
En caso de error en la clase clsCapaX, propaga este error a la clase clsCapaYfija la propiedad NumError

Distribución de un programa de 3 capas

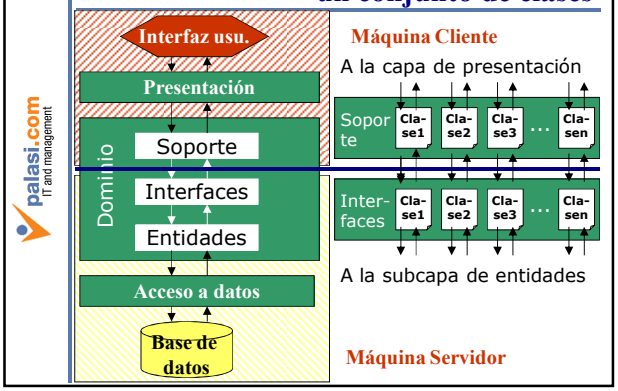
- Hasta ahora, hemos programado como si sólo tuviéramos una sola máquina, pero, como hemos visto, esto no suele ser común en desarrollo en tres capas.
- Debemos saber cómo hacer para distribuir las diferentes subcapas en varias máquinas.
- La dificultad se presenta cuando una subcapa debe de acceder a otra subcapa situada en una máquina diferente.

Mail: palasi@palasi.com Web: www.palasi.com

Una subcapa intenta acceder a otra en una máquina diferente

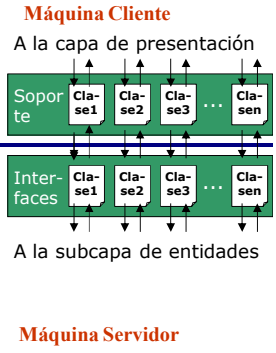


Cada subcapa no es más que un conjunto de clases



El problema se reduce a nivel de clase

- ¿Cómo una clase de Visual Basic accede a otra clase que se encuentra en una máquina diferente?
- Por motivos técnicos, el problema se reduce a la instanciación.



Instanciando un objeto de una clase en la misma máquina

Clase clsX.

```
Option Explicit
...
Dim objY As Object
Set objY = _
 CreateObject("clsY")
...
```

Clase clsY.

```
Option Explicit
...
```

- El CreateObject crea una instancia de la clase clsY.
- Problema: Cómo hacemos para que el CreateObject funcione aunque la clase clsY esté en otra máquina.

CreateObject que instancia una clase de otra máquina (1)

Procedimiento en la máquina servidor

- La clase de la máquina remota (servidor) debe estar compilada como un componente COM.
- Debe habilitarse ese componente para la ejecución remota (con DCOM o COM+). Con DCOM:
  - 1. Por medio del registro.
  - 2. Por el programa de configuración de DCOM, DCOMCNFG.EXE

CreateObject que instancia una clase de otra máquina (2)

Procedimiento en la máquina cliente

- Método 1. Especificar en el CreateObject el nombre o URL de la máquina donde se encuentra la clase.

```
Set objY = CreateObject("clsY",
  "MISERVIDOR")

Set objY = CreateObject("clsY",
  "www.misitio.com")
```


Mail: palasi@palasi.com Web: www.palasi.com

CreateObject que instancia una clase de otra máquina (3)

Procedimiento en la máquina cliente

- Método 2. Dejar el CreateObject como está. Pero indicarle a Windows que esa clase se encuentra en otra máquina.
  Set objY = CreateObject("clsY")
- Esto se indica con DCOM o COM+. Con DCOM:
  - 1. Por medio del registro.
  - 2. Por el programa de configuración de DCOM, DCOMCNFG.EXE

Método 1 contra Método 2

- El método 1 tiene la ventaja que es más directo.
- El método 2 tiene la ventaja de que es más flexible. De hecho, el programa (y el programador) no sabe si la clase es local o remota, sólo Windows (y el administrador) lo saben.

Una observación final

- Para que todo esto funcione con DCOM, cada subcapa debe compilarse como un EXE ActiveX.
- Los DLL ActiveX no funcionan con este método y, en general, no son adecuados para instanciarlos en máquinas diferentes ya que son servidores del mismo proceso que el cliente.

XML

- eXtensible Markup Language: Un lenguaje estándar para describir conjuntos de datos en cadenas de texto con un formato parecido a HTML.
- Sirve para transmitir datos con independencia de la plataforma y el lenguaje. Puede ser útil en algunos proyectos.
- ADO tiene métodos para "traducir" recordsets ADO en cadenas XML y viceversa.

Programa del seminario (y 2)

- 9. Capa de dominio: Clases de Soporte.
- 10. Capa de presentación.
- 11. Otros aspectos.
- 12. Conclusión.

Conclusión

- La programación en 3 capas incrementa la calidad y la flexibilidad de una aplicación, así como la posibilidad de ejecutarla por Internet.
- En este curso hemos visto los conceptos básicos y cómo se aplican a un caso real.
- A partir de aquí, se puede incrementar el dominio mediante práctica.

Mail: palasi@palasi.com Web: www.palasi.com