UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

Autores: Milagros Pacheco Castañeda

Jesús Romero Martínez

Programación con lenguajes de cuarta generación		Clave: 1267
Plan: 2005		Créditos: 8
Licenciatura: Informática		Semestre: 2
Área: Informática (Desarrollo de Sistemas)		Hrs. Asesoría: 4
Requisitos: **Introducción a la programación, 1er. semestre**		Hrs. por semana: 4
Tipo de asignatura:	Obligatoria (x)	Optativa ( )

Objetivo general de la asignatura

Al finalizar el curso el alumno será capaz de diferenciar las características e identificar las ventajas y desventajas de los lenguajes de cuarta generación mediante su utilización en el Desarrollo de Sistemas.

Temario oficial (64 horas sugeridas)

1. Antecedentes de los lenguajes de cuarta generación. (6 horas)
2. Fundamentos de un lenguaje de cuarta generación. (12 horas)
3. Herramientas de acceso a las bases de datos (12 horas)
4. Módulos de clase de un lenguaje de cuarta generación (12 horas)

5. Active-X y Control-Y Component Object Model (12 horas)

6. Optimización y distribución de aplicaciones (10 horas)

Introducción

> Las herramientas de ayuda al desarrollo surgieron para intentar dar solución a los problemas inherentes a los proyectos de generación de aplicaciones informáticas: plazos y presupuestos incumplidos, insatisfacción del usuario etc. Algunas de estas herramientas se dirigen principalmente a mejorar la calidad, como es el caso de las herramientas **CASE** *(Computer Aided Software Engineering,* Ingeniería de *Software* Asistida por Ordenador). Otras van dirigidas a mejorar la productividad durante la fase de construcción, como es el caso de los lenguajes de cuarta generación (4GL)[1].
>
> No existe consenso sobre lo que es un *lenguaje de cuarta generación* (4GL Fourth Generation Language). Los lenguajes de cuarta generación suponen una evolución de los de tercera generación. Los lenguajes de cuarta generación son más fáciles de usar que los 3GL: suelen incluir interfaces gráficos y capacidades de gestión avanzadas, pero consumen muchos más recursos de la computadora que la generación de lenguajes previa.[2]

En esta asignatura el estudiante conocerá las características, los fundamentos, las capacidades y los alcances de los lenguajes de cuarta generación, revisará alguna en particular y aprenderá a utilizar el entorno de desarrollo que ofrecen para programar. La asignatura se desglosa en seis temas que se describen brevemente a continuación.

En el **tema I** se revisa los principios de los lenguajes de programación como sintaxis, semántica y gestión de memoria. Se describen los paradigmas de programación que permiten comprender la filosofía de los lenguajes y por último,

---

[1] Material en línea, “Herramientas CASE”, disponible en: http://html.rincondelvago.com/herramientas-case.html, recuperado el 18/02/09.

[2] Master Magazine: “Definición de 4GL”, en línea, disponible en: http://www.mastermagazine.info/termino/3671.php, recuperado el 18/02/09.

se repasa la historia de los lenguajes de programación para entender la naturaleza de cada uno, identificar las similitudes y diferencias que comparten, y poder definir qué es un lenguaje de Cuarta Generación.

En el **tema 2** se utiliza el lenguaje de programación Visual Basic y se describe los elementos que forman parte de su entorno de desarrollo, el tipo de aplicaciones que se pueden generar, la distribución de herramientas para la generación de programas y los elementos de sintaxis, variables constantes y tipos de datos a utilizar propios del lenguaje.

En el **tema 3** se describe la tecnología OleDB utilizada para conectarse a una base de datos y las herramientas de manipulación de datos que permiten generar aplicaciones en Windows.

Se describe la tecnología basada en controles ActiveX para la conexión y manipulación de datos conocida como ADO y se enumeran las herramientas que Visual Basic proporciona para trabajar con las bases de datos.

En el **tema 4** se describe el concepto de clases, base principal de la programación orientada a objetos, y aprovechando lo aprendido en el tema anterior, se revisará cómo conectar una clase creada por el programador con una base de datos para explotar el manejo de la información con clases.

En la **tema 5** se revisa la tecnología COM (Component Object Model) y Control ActiveX para aprender a utilizar el uso de objetos en la creación de aplicaciones. Se describen las ventajas de utilizar estas tecnologías en el diseño de programas.

En la **tema 6** se describen las técnicas de depuración que se pueden utilizar para desarrollar aplicaciones más eficientes y se muestra cómo poder generar un

programa de instalación para distribuir las aplicaciones realizadas por los programadores.

TEMA 1. ANTECEDENTES DE LOS LENGUAJES DE CUARTA GENERACIÓN

Objetivo particular

El alumno definirá un lenguaje de programación conociendo los principios de diseño de éstos, además clasificará los lenguajes por su evolución e identificará las ventajas y desventajas a partir de sus cualidades.

Temario detallado

1.1 Contenido

1.2 Historia

1.3 Definición

Introducción

Desde el inicio de la humanidad se han creado lenguajes que permitan transmitir ideas, pensamientos, información. Si consideramos a todo aquello que nos permite expresarnos como un lenguaje podemos definir de forma amplia a un lenguaje como el **conjunto de reglas que nos permiten construir frases para expresar o transmitir mensajes**.

Por otra parte tenemos a la computadora como “Una herramienta capaz de manejar grandes cantidades de datos a gran velocidad”. Conjuntando estas dos herramientas en el campo de la computación tenemos ahora al ser humano con la necesidad de definir lenguajes para comunicarse con una computadora. A estos

lenguajes les llamamos lenguajes de programación. Una definición aceptable de lenguajes de programación es: "*Son el medio de expresión por el cual se construyen los sistemas de cómputo*", o "*Conjunto de reglas que nos permiten representar en una computadora una idea*".

Los lenguajes de programación difieren de los naturales debido a que tienen un dominio de expresión limitado, su dominio es el conjunto de algoritmos.

En este tema se revisará la historia de los lenguajes de programación para entender la naturaleza, similitudes y diferencias, hasta llegar a los lenguajes de cuarta generación.

1.1 Contenido

Los lenguajes de programación están diseñados para facilitar la comunicación de ideas, sin embargo, estas ideas son de tipo algorítmico y la comunicación se entabla entre personas y equipos de cómputo. Para diseñar un lenguaje de programación se toman en cuenta ciertas características que colaboren en la comprensión de cómo trabajan los lenguajes de programación.

Principios del diseño de lenguajes

Debido a la diversidad de problemas o modelos que encontramos en el mundo real se crean, complementan o transforman los lenguajes de programación tratando de decir de forma más fácil lo que no se puede decir con otros.

Simplificando el desarrollo de sistemas de cómputo, podemos decir que este consiste en hacer una abstracción del mundo real bajo algún paradigma[3] y

[3] "Ejemplo que sirve de norma". Diccionario Larousse 2004

representarlo en un equipo de cómputo por medio de un lenguaje de programación.

Figura 1.1 Desarrollo de sistemas

Debido a que en el mundo real encontramos un número infinito de problemas con diferentes características, es necesaria la existencia de diferentes tipos de lenguajes de programación que permitan la expresión de algoritmos de la mejor manera. Sin embargo todo lenguaje de programación tiene los principios de diseño, los cuales podemos agrupar en las categorías:

- Sintaxis
- Sistemas de tipos de datos y semántica
- Gestión de memoria

Sintaxis

La sintaxis de un lenguaje de programación es la definición de lo que constituye un programa gramaticalmente válido y se define por una gramática. Formalmente podemos definir un lenguaje haciendo uso de la clasificación de Chomsky (Chomsky, 1957) de la siguiente forma.

Una gramática generativa G = (V_N, V_T, S, F) se dice que es de tipo *i* si satisface las restricciones correspondientes:

I=0 : Sin Restricciones

I=1 : Toda regla de reescritura en F de la Forma $Q_1AQ_2 \rightarrow Q_1PQ_2$, con Q_1, Q_2 y P en $(V_N \cup V_T)^*$, $A \in V_N$, y $P \neq \lambda$, excepto la regla $S \rightarrow \lambda$, la cual puede estar en F, en este caso S no puede aparecer en el lado derecho de otras reglas de producción.

I=2 : Si toda regla en F tiene la forma $A \rightarrow P$ donde $A \in V_N$ y $P \in (V_N \cup V_T)^*$

I=3 : Si toda regla en F tiene la forma $A \rightarrow PB$ o $A \rightarrow P$, donde $A,B \in V_N$ y $P \in V_T^*$

Un lenguaje es de tipo *i* si es generado por una gramática de tipo *i*. La clase o familia de lenguajes de tipo *i* se denota por $£_i$.

Las gramáticas de tipo 0 son llamadas gramáticas estructuradas por frase. Las gramáticas de tipo 1 son llamadas sensibles al contexto. Las gramáticas de tipo 2 son llamadas de contexto libre y finalmente las de tipo 3 son llamadas gramáticas regulares.

En G:

V_N: Es el conjunto de símbolos NO terminales, son los símbolos que se derivan o reescriben.

V_T: Es el conjunto de símbolos terminales, son los símbolos que no se derivan o reescriben.

S: Es el símbolo inicial, es el *Símbolo no Terminal* a partir del cual se derivan todas las cadenas de lenguajes.

F: Es el conjunto de reglas de producción, indican las derivaciones que se pueden hacer, su forma es <lado izquierdo> → <lado Derecho>, y se lee: el <lado izquierdo> se deriva o reescribe en <lado Derecho>

En este aspecto, por ejemplo, se busca que una gramática no sea ambigua, es decir que podamos generar una cadena con dos o más derivaciones diferentes. A demás se pueden estudiar las características de los lenguajes formalmente, el tipo

de propiedades que cumplen (cerradura, distributividad, conmutatividad, etc.), las operaciones que pueden realizar entre ellos (unión, intersección, complemento, resta).

Sistemas de tipos de datos y semántica

Los sistemas de tipos de datos son muy importantes en el diseño de lenguajes de programación ya que se utilizan para formalizar la definición de los tipos de datos y su utilización correcta en los programas. Generalmente se asocia con la sintaxis, por lo que se pueden considerar como una extensión de definiciones que impone restricciones sintácticas específicas que no pueden expresarse con una gramática. El sistema de tipo de datos se encuentra entre la sintaxis y la semántica y podemos visualizarlo apropiadamente en cualquiera de los dos.

Un tipo de datos es un conjunto de valores (dominio) y un conjunto de operaciones sobre los valores (funciones). Por ejemplo tenemos el conjunto de los enteros (Z) como conjunto de valores y las operaciones de +,-,*,/ para los enteros. Para el tipo boolean tenemos los valores {true, false} y las operaciones and, or y not.

En el sistema de tipos se asocian variables con tipos. Por ejemplo los lenguajes C, Pascal, Modula, son lenguajes fuertemente tipificados. Donde el tipo de dato de una variable no cambia a lo largo de la ejecución del programa. Los lenguajes débilmente tipificados permiten que el tipo de dato de una variable cambie a lo largo de la ejecución del programa, Lisp y VB 6.0 son un ejemplo.

“La semántica de un lenguaje de programación es una definición del significado de cualquier programa que sea sintácticamente válido” (Tucker, 2003).

La semántica puede ser:

- Operacional: Que consiste en una descripción precisa de lo que ocurre cuando se ejecuta el programa. Proporciona una definición del significado del programa simulando el comportamiento del mismo en un modelo de

equipo que tenga un conjunto de instrucciones básicas y una unidad de memoria.

- Axiomática: Es una descripción formal que permite demostrar que el programa es correcto y completo. Se basa en un sistema riguroso y lógico que permite ver a un programa como una función f: N → N. Es útil en la exploración de las propiedades formales de los lenguajes.

Gestión de memoria

Como parte de la definición del lenguaje se encuentra la forma, los algoritmos para el uso y manejo de la memoria, que comprende la correspondencia entre las estructuras de datos y la memoria. El manejo de variables estáticas y dinámicas. El tiempo de vida de las variables, definiendo en qué momento se crean y se destruyen. Si se cuenta con recolector de basura o no entre otras características.

Paradigmas de la programación y dominios de aplicación

Aunque existen lenguajes de programación en los que podemos decir mucho algorítmicamente, que cubren varios dominios de aplicación, no podemos decir todo de una manera fácil con ellos. Por ejemplo: Si tratamos de desarrollar una aplicación que maneje números complejos (A+Bi), seguramente lo podemos escribir en cualquier lenguaje. Sin embargo, es más fácil en *Fortran* ya que tiene el tipo dato *Complejo* como tipo de dato primitivo. También tenemos paradigmas de programación (modelos o patrones a seguir en la construcción de software) que nos permiten expresar con mayor facilidad ciertas situaciones.

Los paradigmas de la programación son:

- **Programación Imperativa**: Todo programa es una secuencia finita, pasos o instrucciones que la computadora ejecuta. El programa toma los datos de entrada, los procesa ejecutando línea a línea y da una salida. Es imperativa

porque se dice en el programa qué se debe hacer en cada momento. Los ejemplos típicos son Cobol, Fortran, C++.

- **Programación Orientada a Objetos**: El programa es una colección de objetos, cada objeto y tiene propiedades y métodos que definen su comportamiento. Los objetos interactúan entre sí por medio de mensajes y transformando su estado. Los ejemplos clásicos son Smalltalk, Java, C++ e Eiffel.

- **Programación funcional**: El programa es una colección de funciones f:X→Y, las cuales por medio de composición de funciones, condicionales y recursividad permiten representar algorítmicamente una solución. Los lenguajes más representativos son: Lips, Écheme, Haskell y Ml.

- **Programación lógica o declarativa**: El programa es una colección de enunciados basados en lógica de primer orden y cálculo de predicados que se centran en describir *¿Qué se debe hacer?,* más que en *¿Cómo se debe hacer?* Y es tarea de la máquina de inferencia ejecutar el programa revisando o buscando en la base de conocimiento los valores que deben tomar las variables para que los enunciados sean verdaderos. El lenguaje de programación más importante es Prolog.

- **Programación guiada por eventos**: El programa se encuentra en un ciclo respondiendo a eventos generados en un orden no predecible. Estos eventos se generan por acciones del usuario, los más comunes son clic del Mouse, presionar una tecla. Visual Basic y Java son ejemplo de este tipo de lenguajes.

- **Programación concurrente**: El programa es una colección de proceso que interactúan entre sí para compartir recursos. Pero que operan

asíncronamente. Esto es, que son ejecutados por el mismo procesador.

Algunos lenguajes están pensados para soportar más de un paradigma, tal es el caso de C++, que es imperativo y orientado a objetos.

1.2 Historia

Recordando un poco la historia de la computación encontramos que máquinas como la ENIAC que se programaba por medio de interruptores, el programa formaba parte del hardware, ahí ya se contaba con un lenguaje de programación basado en 1's y 0's, propio de la máquina. Estos lenguajes pertenecen a los lenguajes de bajo nivel. Tiempo después con el modelo de John von Newmann donde se propone almacenar programas y datos en la misma área de memoria se tiene un gran avance al permitir cambiar rápidamente el programa o instrucciones de la computadora pero aun se debía conocer y programar en lenguaje de la máquina, la EDVAC es la primera computadora que adopta este modelo. Posteriormente comienzan a surgir lenguajes de programación de alto nivel, en los que ya no es necesario conocer el lenguaje de máquina o de bajo nivel. Se puede programar en estos lenguajes y con ayuda de un traductor las instrucciones se pasan a código de máquina. Estos traductores son programas de computadora que reciben un programa en código fuente (alto nivel) y lo transforman o reescriben en código objeto (bajo nivel), dependiendo de la forma en que funcionan los podemos clasificar como *Compilador o Intérprete*.

La mayoría de los lenguajes tienen una utilidad limitada y su uso va cambiando con el paso del tiempo ya que surgen nuevos lenguajes, paradigmas, equipos de cómputo, etc. En general, nuevas formas de hacer software. Algunos de ellos han servido para desarrollar nuevos lenguajes, otros se quedan en los laboratorios, tal es el caso del lenguaje B que fue desarrollado por Ken Thompson en 1970 en los laboratorios Bell de AT&T e inspirado en BCPL de Martin Richard. A su vez sirve

para que en 1972 Dennis Ritchie lo modifique y escriba C. Al parecer C es un lenguaje que llegó para quedarse.

Son notables Cobol y Fortran que nacieron en la década de los 50. Pensados para correr en equipos grandes (Mainframe). En alguna época (los 80) fuero pensados para los equipos pequeños. Pero han evolucionado y se mantienen hoy en día.

Una de las características de los nuevos lenguajes respecto a sus antecesores es que incluyen clases (en el caso de los OO) o bibliotecas (en los otros casos) que contiene un número considerado de objetos, funciones que le permiten al programador ahorrar tiempo de desarrollo al contar con los algoritmos ya programados, optimizados e implementados. Hoy en día conocer el lenguaje es conocer este conjunto de clases o funciones.

Generaciones de los lenguajes de programación

Los lenguajes de programación los podemos clasificar en generaciones, de acuerdo con sus características:

- **Primera generación**: Manejaban 0's y 1's, se programaban con tarjetas perforadas. Los programas eran rápidos. No se tenía portabilidad y son difíciles de aprender.
- **Segunda Generación**: Lenguaje ensamblador con el uso de etiquetas y saltos. Más abstractos que el lenguaje de máquina y eficientes. Con la desventaja que es propia de la máquina.
- **Tercera Generación**: Lenguajes de alto nivel, que soportan a la programación estructurada, con un mayor nivel de abstracción y funcionalidad que sus antecesores. Pero menor rendimiento.
- **Cuarta Generación**: Son lenguajes orientados a resolver problemas, reducir el costo y esfuerzo, fáciles de aprender a cambio de tener una aplicación limitada.

Cualidades de un lenguaje de programación

Es difícil decir cuáles son las cualidades que debe tener el lenguaje ideal debido a la extensa gama de áreas de aplicación. Es decir, no podemos tener un lenguaje que sea óptimo para todas las áreas. Es análogo a pensar que existe un carro perfecto, bueno, lujoso, barato, espacioso, compacto, ahorrador, etc. Por eso nos limitaremos a mencionar las características deseables en un lenguaje de programación.

Simplicidad: Se refiere al grado de dificultad para el programador para escribir programas y leer programas de otros, aprender y enseñar el lenguaje. Hay lenguajes como C que sacrifican la simplicidad y claridad a cambio de la velocidad y eficiencia. Por eso se dice que "*C es un lenguaje para programadores expertos*". Así también encontramos a Pascal que es un lenguaje amigable para enseñar la programación estructurada.

Uniones: Se refiere al momento en que se define el tipo de dato de las variables. Es decir, en un lenguaje fuertemente tipificado la definición del tipo de una variable se puede hacer en tiempo de compilación. Por el contrario en los débilmente tipificados se puede hacer en tiempo de ejecución. Esto puede ser bueno o malo ya que permitir que una variable cambie de tipo en tiempo de ejecución permite errores "lógicos", por otra parte en algunos casos son buenos cuando se desconoce el tipo de objeto que regresará una función.

Ortogonalidad: Se refiere al significado de las palabras reservadas o símbolos. Si una palabra reservada siempre tiene el mismo significado independientemente del contexto en que se use el lenguaje tiene mayor ortogonalidad que en otro donde no sucede eso. Un ejemplo es el significado de "+" si siempre representa la suma aritmética es ortogonal, por el contrario si representa a la suma para variables numéricas y concatenación para variables de tipo cadena entonces es menos ortogonal.

Fiabilidad de los programas: Se refiere al comportamiento del programa, al momento de cambiarlo de plataforma, de ejecutarlo más de una vez con el mismo conjunto de datos de entrada. Un lenguaje es más fiable si es fuertemente tipificado, restringe el uso de *alias* y las pérdidas de memoria y si tiene una sintaxis y semántica bien definida.

Abstracción: Se refiere al nivel de complejidad de las funciones o clases incorporadas que le ahorran al programador tiempo de desarrollo al proporcionarle algoritmos ya implementados. Por ejemplo: en Pascal se cuenta con el tipo de datos *string* que nos permite almacenar cadenas y algunas funciones sobre cadenas como: *len(), upper(), etc.* Pero si queremos implementar alguna estructura abstracta como una pila, cola, etc. Debemos definir a la estructura y las funciones para manejarla. Hoy en día la mayor parte de los lenguajes ya cuenta con una clase *Pila* que tiene los métodos para manejarla. Así el programador únicamente debe indicar que utilizará un objeto de la clase *Pila* para contar con todos los algoritmos ya programados.

Implementación eficiente: Se refiere a la facilidad o dificultad que se tiene para implementar prácticamente un lenguaje de programación. Podemos tener un nuevo paradigma con un nuevo lenguaje, pero no se cuenta con los medios necesarios para implementarlo eficientemente. Tal fue el caso de Smalltalk que fue el primer lenguaje orientado a objetos, pero debido al tipo de computadoras y sistemas operativos que existían en ese tiempo no era posible tener un compilador eficiente.

En resumen todo lenguaje tiene características que lo hacen fuerte y otras que lo debilitan.

1.3 Definición

Los Lenguajes de Cuarta Generación se pueden definir como entornos de desarrollo de aplicaciones apoyados por una serie de herramientas de alto nivel.
Los 4GL contienen una sintaxis distinta para la representación del control y las estructuras de datos con un mayor nivel de abstracción. Presentan ciertas características que los distinguen de los demás lenguajes de programación.

Características de un 4GL

- Es un lenguaje no procedimental [*non-procedural*]. Solo se define **qué se debe hacer**, no **cómo se debe hacer**.
- Se apoya en herramientas de alto nivel denominadas herramientas de cuarta generación que contienen los algoritmos necesarios para decir cómo hacer lo que el usuario necesita.
- Es limitado el tipo de problemas que pueden resolver.
- Permite el manejo y manipulación de datos basado en el lenguaje SQL (*Structured Query Language*)
- Combinan características procedimentales (Permite especificar condiciones con sus respectivas acciones) y no procedimentales (Pide que se indique el resultado deseado).
- Aumento de productividad por la utilización de funciones preprogramadas.
- El entorno de desarrollo facilita la realización de determinadas tareas como diseño de pantallas o informes.

Categorías de los 4GL

- Lenguajes de presentación, como lenguajes de consultas y generadores de informes.
- Lenguajes especializados, como hojas de cálculo y lenguajes de bases de datos.
- Generadores de aplicaciones que definen, insertan, actualizan y obtienen datos de la base de datos.

- Lenguajes de muy alto nivel que se utilizan para generar el código de la aplicación.

Clasificación de los 4GL

1. Por su relación con un manejador de base de datos

    - **Lenguajes ligados a una base de datos.** Son lenguajes propietarios, lo que quiere decir que sirven únicamente para acceder a esa base de datos en particular. El aprovechamiento de los recursos del manejador es muy alto.
    - **Lenguajes Independientes del manejador de base de datos.** Permiten acceder a diferentes bases de datos, generalmente aquéllas que soportan un estándar común.

2. Por la naturaleza de su sintaxis:

    - **Lenguajes procedimentales.** El programa se desarrolla como una secuencia de pasos que la computadora ejecuta para llegar al fin deseado.
    - **Lenguajes conducidos por eventos.** Permiten especificar la ejecución de rutinas asociadas con acciones dadas por el usuario, tales como apretar una tecla o mover el ratón, sin tener que codificar cada paso dado para ejecutar dicha acción.

Son ejemplos de 4GL

- SQL
- QBE
- Visual Basic.
- Lenguajes de soporte a la toma de decisiones
- Los lenguajes de prototipos
- Lenguajes de especificación formal

- SheerPower4GL
- PowerBuilder
- WinDev
- Focus
- Natural
- Progress4GL
- Oracle Reports
- PostScript
- MatLab
- Ramis, entre otros

Para utilizar un 4GL se debe tener claro qué se debe hacer, cambios en la forma normal de hacer software. Para esto, se debe ser analista, dominando técnicas estructuradas, conceptos de diseño de interfaz gráfica, conceptos de arquitectura, conceptos de orientación a objetos y de principios de diseño. Y todo esto para poder obtener una mayor productividad, una mayor facilidad al dar mantenimiento y además una mejor apariencia de la aplicación.

Bibliografía del tema 1

Tucker Allen, y Robert Noonan. (2003) *Lenguajes de Programación principios y paradigmas*, México, McGraw Hill.

RÉVÉSZ György E., (1991), *Introduction to formal languages,* Dover Publications.

Actividades de aprendizaje

A.1.1 Con la finalidad de que identifiques las características de diversos lenguajes, realiza una investigación y completa el siguiente cuadro.

Lenguaje	Año	Autor	Generación	Paradigma de programación
Fortran				
Fortran 66				
Fortran 77				
Fortran 90				
Fortran 97				
Cobol				
Cobol 68				
Cobol 74				
Cobol 85				
Algol 60				
Algol 68				
Pascal				
Modula 2				
Ada				
Ada 95				
Bcpl				
C				
C ++				
C #				
Smalltalk 80				
Eiffel				
Java				

Lenguaje	Año	Autor	Generación	Paradigma de programación
Lips				
Schema				
Haskell				
Prolog				
Clp				
Sequel				
Sql92				
Visual Basic				
Pascal concurrente				
Occam				
Csp				

A.1.2. A partir de la tabla anterior, indica cuál es el tipo de semántica de los siguientes lenguajes de programación:

1. Lenguaje C
2. Visual Basic 6

Cuestionario de autoevaluación

1. ¿Por qué es necesaria la creación de diferentes lenguajes de programación?
2. Indica la diferencia entre el lenguaje natural y los lenguajes de programación
3. ¿Qué entiendes por sintaxis?
4. ¿Qué entiendes por semántica?
5. ¿Qué es una gramática generativa?
6. ¿En qué año se crea “C” y quién es su creador?
7. ¿Cuáles son los paradigmas de la programación? Descríbelos brevemente.

8. Enuncia y describe las cualidades deseables de un lenguaje de programación.
9. ¿Cuáles son las características de un 4GL?
10. ¿Cuáles son los tipos de 4GL que existen?

Examen de autoevaluación

Elige la opción correcta

1. ¿Qué niveles tenemos en la clasificación de Chomsky?
    a) Lenguajes regulares, de contexto libre, sensibles al contexto, estructurados por frase.
    b) Lenguajes no regulares, de contexto libre, sensibles al contexto, estructurados por frase.
    c) Lenguajes regulares, contexto libre, sensibles al contexto, sin restricciones.
    d) Lenguajes regulares, libres de contexto, sensibles al contexto, estructurados por frase.
    e) Ninguna de las anteriores

2. Si un programa es sintácticamente correcto podemos decir:
    a) El programa está bien escrito y es lógicamente correcto
    b) El programa está bien escrito y tendremos a la salida lo que esperamos
    c) El programa está bien escrito
    d) El programa tiene un problema lógico
    e) El programa está bien escrito pero no va a correr

3. Si un programa es semánticamente correcto podemos decir:

a) El programa está bien escrito y es lógicamente correcto

b) El programa es lógicamente correcto

c) El programa está bien escrito

d) El programa está bien escrito pero no va a correr

e) El programa está en un lenguaje de cuarta generación

4. ¿Son cualidades deseables de un lenguaje de programación?

a) Ortogonalidad, simplicidad, fiabilidad, implementación total, abstracción.

b) Ortogonalidad, cohesión, simplicidad, fiabilidad, implementación total, abstracción.

c) Ortogonalidad, acoplamiento, fiabilidad, implementación total, abstracción.

d) Ortogonalidad, simplicidad, fiabilidad, implementación eficiente, abstracción.

e) Ortogonalidad, simplicidad, fiabilidad, implementación eficiente, abstracción, alta cohesión.

5. ¿A qué se refiere la ortogonalidad?

a) Al significado de las palabras reservadas

b) Al significado de las palabras no reservadas y reservadas

c) Toda palabra reservada siempre tiene el mismo significado independientemente del contexto

d) Al contexto de las palabras reservadas y no reservadas

e) Toda palabra tiene un significado independiente del contexto

6. ¿A qué se refiere la fiabilidad de un programa?

a) El programa siempre se comportará de la misma forma al cambiarlo de plataforma, ejecutarlo más de una vez con los mismos datos

b) El programa puede dar resultados diferentes con los mismos datos de entrada

c) El programa debe ser portable

d) Todo programa debe correr en cualquier plataforma

e) Ninguna de las anteriores

7. ¿Qué pasa con los lenguajes de 4ª Generación?

a) Manejan 0's y 1's, son muy rápidos y difíciles de aprender

b) Son leguajes de alto nivel, difíciles de aprender

c) Son lenguajes fáciles de aprender, orientados a resolver problemas, reducir el costo y esfuerzo. Se pueden utilizar para todo tipo de aplicaciones

d) Son lenguajes fáciles de aprender, orientados a resolver problemas, reducir el costo y esfuerzo. De todos los lenguajes son los más rápidos y eficientes

e) Son lenguajes fáciles de aprender, orientados a resolver problemas, reducir el costo y esfuerzo. El programador dice que 'qué' no 'cómo'.

8. ¿Cuáles son considerados lenguajes de 4ª generación?

a) C, pascal, Visual Basic

b) C++, modula, Prolog

c) Visual Basic, Prolog, Java

d) Sql, Visual Basic, Fortran

e) Sql, QBE

9. ¿C++ es un lenguaje?

a) De 4ta generación

b) Orientado a la programación concurrente

c) Orientado a objetos e imperativo

d) Orientado a objetos

e) Ninguna de las anteriores

10. Para que un lenguaje sea considerado como estructurado ¿qué debe tener?

a) Una estructura de asignación, una de selección y una de repetición

b) Una estructura de asignación, una de selección y una de repetición y el salto incondicional

c) Una estructura de asignación, una de selección y una de repetición

d) Una estructura de asignación, dos o mas de selección y una de repetición

e) El goto, if … then ..else, do..while.

TEMA 2. FUNDAMENTOS DE UN LENGUAJE DE CUARTA GENERACIÓN

Objetivo particular

El alumno reconocerá las características de los lenguajes de cuarta generación y en particular identificará el entorno de desarrollo de un lenguaje de cuarta generación.

Temario detallado

2.1 Entorno de desarrollo Visual Basic

2.2 Creación de una aplicación

Introducción

En este tema se puede revisar cualquier lenguaje de programación resaltando las características que lo hacen pertenecer a la cuarta generación. Sin embargo se toma al lenguaje Visual Basic debido a que cumple con varias características que lo hacen ser un lenguaje de cuarta generación.

Entre las características que podemos citar son las siguientes:

- Cuenta con un entorno de programación que permite rápidamente construir prototipos.
- La generación de ventanas se resume en arrastrar un objeto ventana al proyecto.
- Cuenta con herramientas que nos permiten crear aplicaciones sencillas de acceso a bases de datos sin escribir líneas de código.

2.1. Entorno de desarrollo Visual Basic

Cuando se comienza a trabajar en el entorno de Visual Basic 6.0 (VB), lo primero que debemos seleccionar es el tipo de proyecto que queremos construir. Este proyecto estará formado por todos los archivos que formarán una aplicación. VB como una herramienta CASE nos permite construir:

- Programas estándar (standard EXE)
- Controles active X
- Bibliotecas active X
- Aplicaciones IIS (aplicaciones web)
- Aplicaciones con acceso a bases de datos

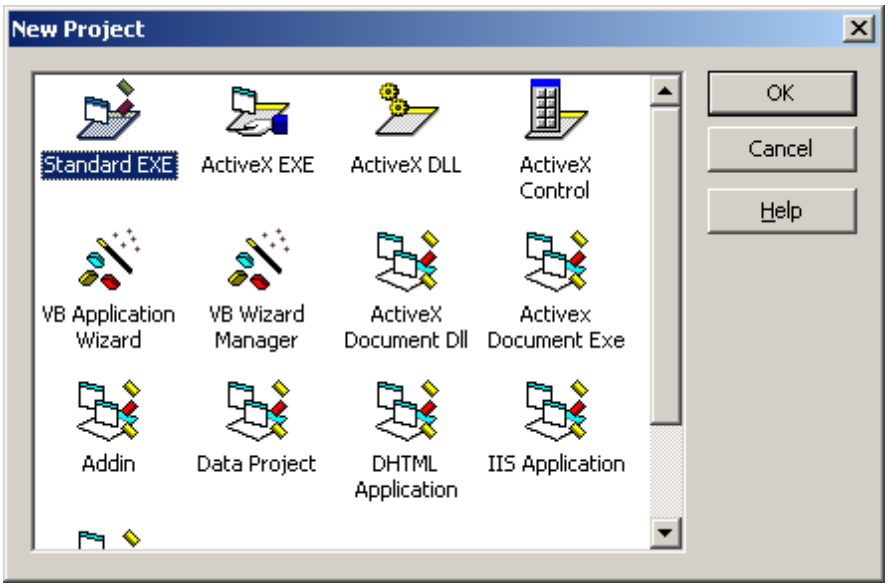


Figura 2.1. Ventana de selección de tipo de proyecto

Una vez que seleccionamos el tipo de proyecto tenemos la ventana del entorno de desarrollo (figura 2), donde encontramos:

1. La *barra de títulos*, la *barra de menús* y la *barra de herramientas*
2. La *caja de herramientas* (*toolbox*)
3. La zona de trabajo, donde se encuentran los *Formularios* (*form*) donde podemos ir poniendo controles

4. La ventana de *proyecto*, que muestra los formularios y otros módulos de programas que forman parte de la aplicación.
5. La ventana de *Propiedades*, en la que se pueden ver las propiedades del objeto seleccionado.
6. La ventana *FormLayout*, que permite determinar la forma en que se abrirá la aplicación cuando inicie.

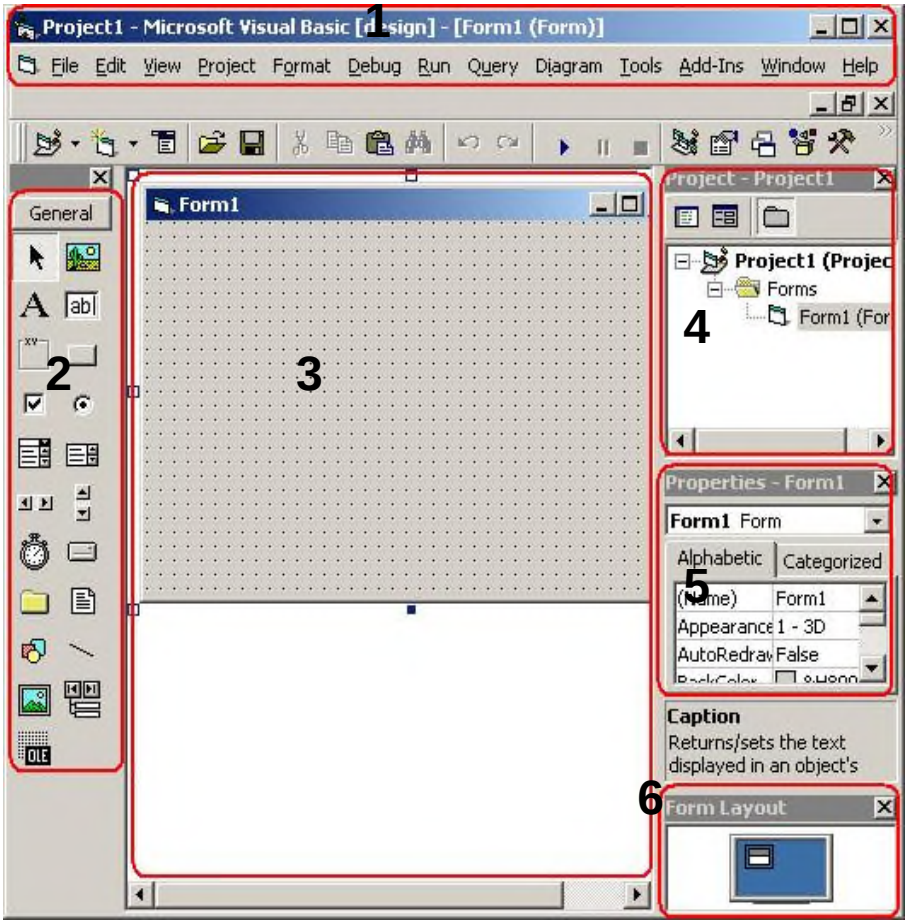


Figura 2.2. Ventana del entorno de desarrollo

Existen otras ventanas para edición de código (*Code Editor*) y para observar el contenido o valor de las variables en tiempo de ejecución por medio del *depurador* o *Debugger* (ventanas *Immediate*, *locals* y *watch*). A este conjunto de herramientas y de ventanas se llama *entorno integrado de desarrollo* o IDE (*Integrated Development Environment*).

Visual Basic puede hacer uso de todos los elementos que caracterizan a los programas de *Windows* por utilizar el conjunto de clases que forman parte del sistema operativo *Windows fundation class*. El entorno de *Visual Basic* es sencillo, lógico y natural.

La barra de menú y las barras de herramientas

La *barra de menú* es similar a la de cualquier otra aplicación de Windows (figura 3), y debajo aparecen las *barras de herramientas.* Existen cuatro barras de herramientas: *Debug*, *Edit*, *Form Editor* y *Standard*. Por default sólo aparece la barra *Standard* ya que agrupa las funciones más importantes para trabajar con los proyectos o formularios.

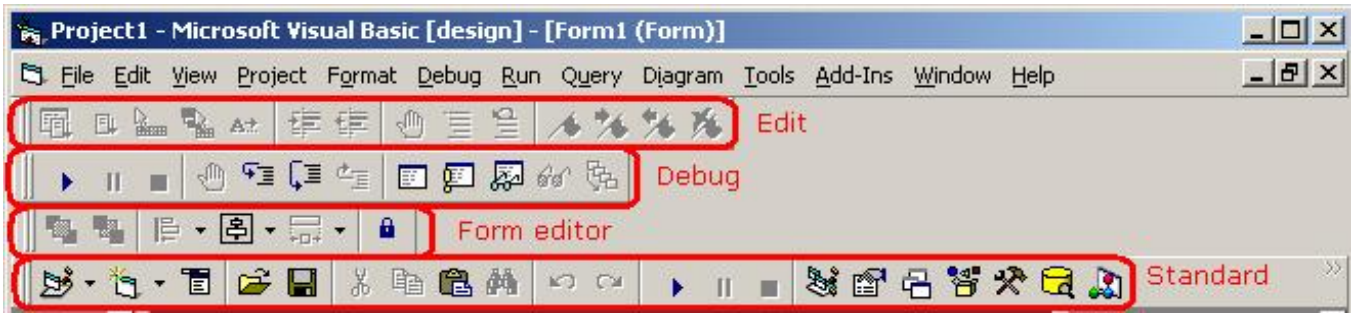


Figura 2.3. Barras de menú

Como en otras aplicaciones de *Windows* se pueden modificar las barras añadiendo o eliminando botones (opción *Customize*).

Las opciones de la barra de menús y funciones principales son las siguientes:

- Menú *File*

  Agrupa las funciones de guardar, guardar como y hace la distinción entre *proyectos* y *archivos* que forman al proyecto. Con los comandos *Open Project* o *New Project* se abre o se crea un nuevo proyecto. Con el

comando *Make ProjectName.exe* se generan los archivos ejecutables de los proyectos.

- Menú *Edit.*

*C*ontiene las funciones de cortar, pegar, buscar, reemplazar, seleccionar, entre otras.

- Menú *View.*

Permite desplegar en pantalla las distintas ventanas del entorno de desarrollo, como son *Inmediate Window, Local Window, Watch Window, Toolbox, Object Browser* entre otros.

- Menú *Projec.*

Permite añadir distintos tipos de elementos como un formulario, un módulo, un módulo de clase, una página de propiedades a un proyecto. Con la opción *Project/Properties* se puede modificar los atributos principales del tipo de proyecto y determinar el formulario principal con el que la aplicación iniciará su ejecución (*Startup Object*).

Dos opciones importantes de este menú es el comando *Components* que permite añadir nuevos controles a la caja de herramientas y el comando *References* que permite agregar librerías (archivos .dll) para su uso en el desarrollo de una aplicación

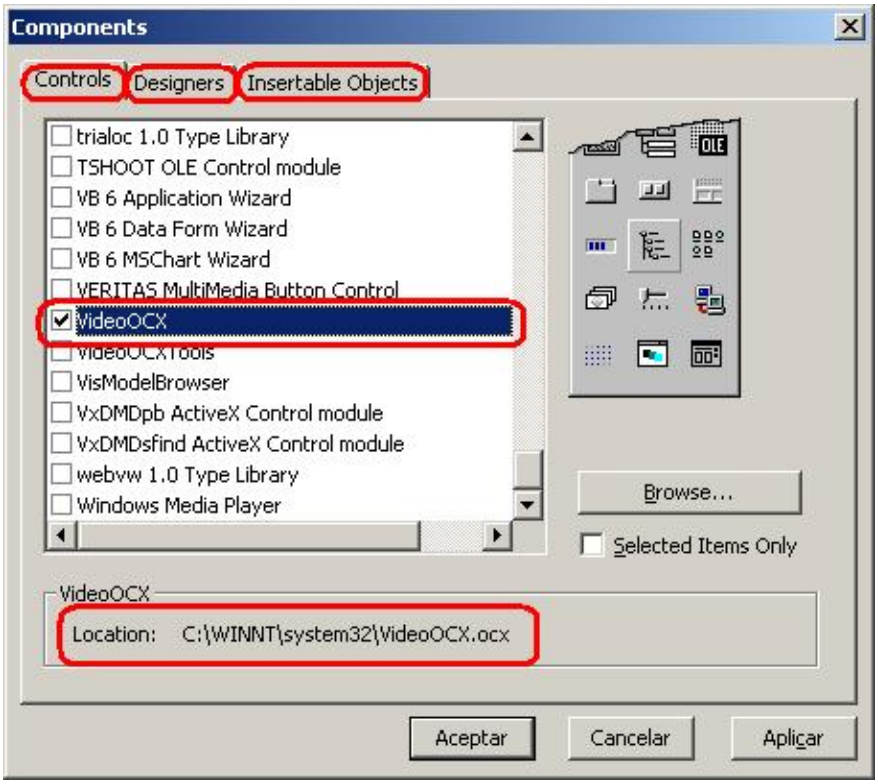


Figura 2.4. Ventana para agregar componentes

- Menú *Format*

Contiene funciones como *Aling, Make Same Size, Horizontal Spacing, Vertical Spacing, Center in Form* que ayudan a especificar la apariencia de la aplicación.

- Menú *Debug*

Permite monitorear la ejecución de una aplicación con sus opciones *StepInto, StepOver;* colocar puntos de ruptura *(Break point)* y agregar *inspectores de variables (Add Watch).* Todas estas opciones con el fin de ayudar a depurar un programa.

- Menú *Run*

Entre sus opciones están *Start, Start with Full Compile, Break, End, Restart* que permiten controlar la ejecución de un programa.

- Menú *Tools*

Tiene las opciones *Add Procedure, Procedure Atributes* para agregar procedimientos y manipular sus propiedades. El submenú *Menu Editor* permite crear menús, y la función *Options* proporciona las opciones de configuración del entorno de desarrollo.

- Menú *Help*

Muestra información clasificada por temas (*Contents*), por alfabeto (*Index*) y permite la búsqueda de información sobre algún tema por el nombre (*Search*).

La caja de herramientas (toolbox)

Incluye los *controles* estándar que se pueden utilizar para diseñar las pantallas o ventanas que formarán a la aplicación.

Los controles son:

- PictureBox
- Label
- TextBox
- FrameCommandButton
- CheckBox
- OptionButton
- ComboBox
- ListBox
- HsCrollBar
- VscrollBar
- Timer
- DriveListBox
- Shape
- Line
- Image
- Data
- OLE

Para insertar un control en un formulario hay que dar clic sobre el icono adecuado de la *caja de herramientas* y colocarlo en el formulario con la posición deseada.

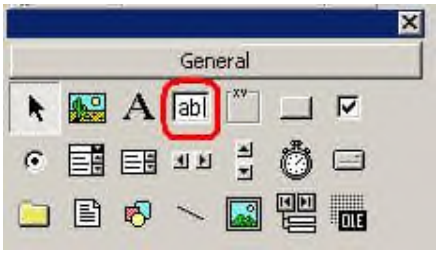


Figura 2.5. Caja de herramientas

Formularios

Los *formularios* son las zonas de trabajo en las que se diseña el programa. En éstas se colocan los controles. Para trabajar con los formularios podemos hacer uso de dos fases, una gráfica donde se colocan los controles y una fase editable que permite escribir sintaxis.

- Fase de diseño *(Form)*

  El formulario se muestra como una rejilla o cuadrícula punteada para colocar los controles de la barra de herramientas. Cuando se ejecuta el programa la cuadrícula no se muestra, sólo se aprecia la ventana donde aparecen los controles.

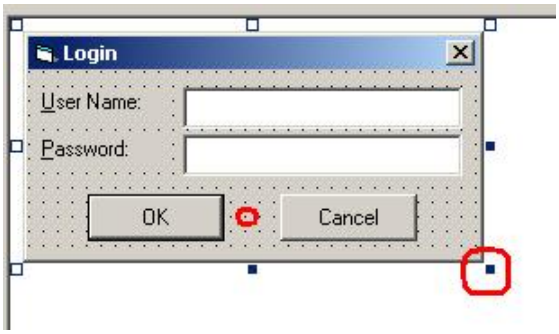


Figura 2.6. Zona de trabajo

- Fase Editor de código *(Code)*

Ventana que permite editar un código de programación que estará escrito en lenguaje *Basic*, y que controlará algunos aspectos del formulario, sobre todo en la forma de reaccionar ante las acciones del usuario (eventos)

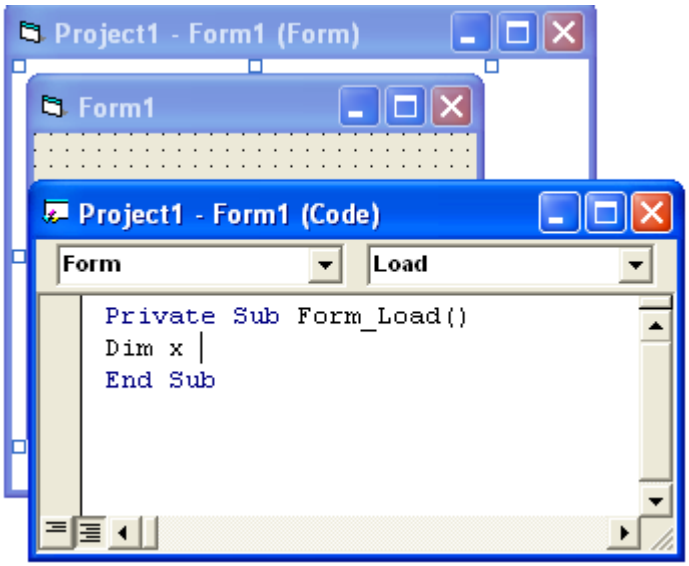


Figura 2.7. Zona de trabajo

El entorno cuenta con varios formularios predefinidos que son *Formas* simples con valores “especiales” en las propiedades y controles agregados que hacen que la forma tenga un comportamiento diferente. Estos formularios realizan tareas comunes de toda aplicación como: *Ventana de login, Ventana de acerca del programa, etc.* Algo interesante es que cuenta con un *Wizard* para crear una ventana de acceso a una base de datos. En pocos pasos, indicando básicamente la ubicación de la base de datos, nombre y password de acceso y seleccionando la tabla o consulta. El *wizard* construye un formulario para poder *agregar, borrar y cambiar los datos*. En este caso no se requiere de ninguna línea de programación para tener un formulario de acceso a los datos de la base de datos.

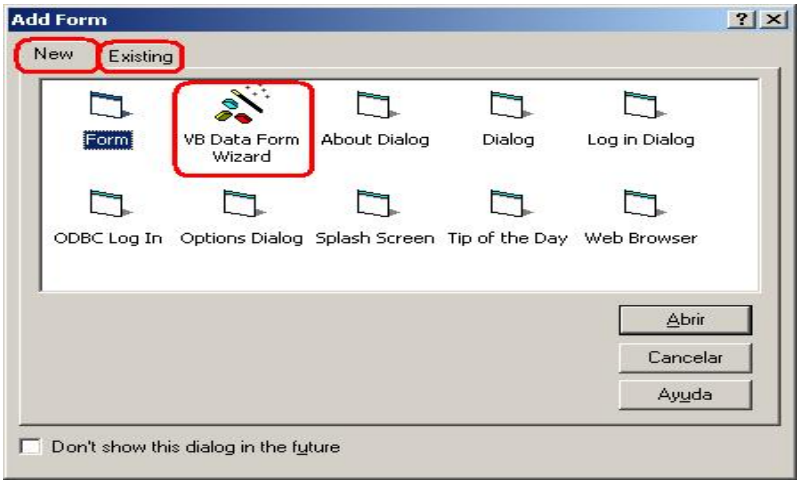


Figura 2.8. Nueva Forma

Una aplicación puede tener varios formularios, pero siempre debemos definir el formulario inicial, el que se abre al correr el programa. Este formulario se determina a partir del menú *Project/Properties*, en *Startup Objects*. En esta ventana se definen las características del proyecto como: tipo de proyecto, formulario de inicio, nombre del proyecto, icono, nombre del ejecutable, asignación de versión, etc.

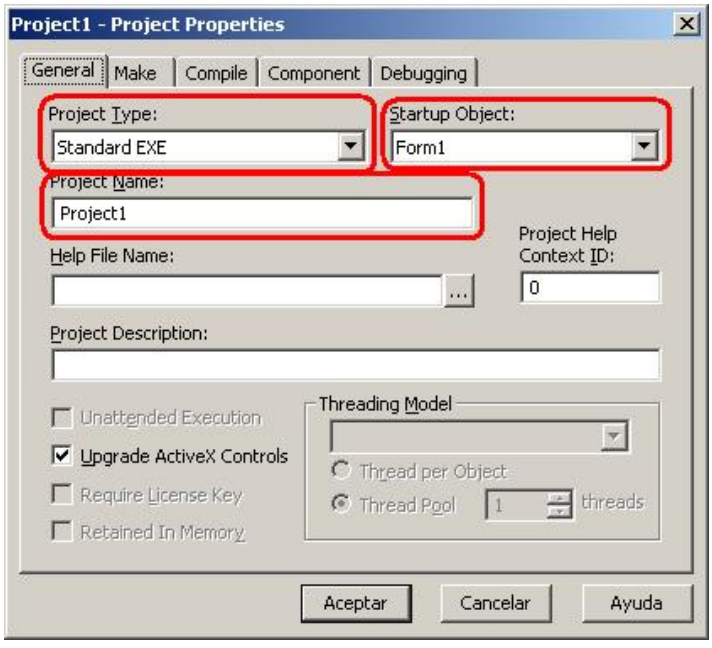


Figura 2.9 Ventana de Propiedades del proyecto

La ventana de proyecto

Esta ventana permite administrar los distintos Objetos (formularios, módulos, etc.) que forman el proyecto. Aquí las dos fases:

- Diseño gráfico de los formularios (utilizando el botón *View Object*)
- Editar el código que contienen (utilizando el botón *View Code*).

Los objetos que se pueden administrar en esta ventana son:

- Formularios (Archivos *.frm)

  Contienen los formularios que componen una aplicación.

- Los *módulos estándar* (ficheros **.bas*)

  Contienen siempre algunas declaraciones de variables globales o *Public*, de procedimientos y funciones que serán accesibles directamente desde todos los formularios.

- Los *módulos de clase* (ficheros **.cls*)

  Contienen clases definidas por el usuario.

- Controles ActiveX *(User control)*

  Contiene las plantillas con el diseño de los controles ActiveX que se generan.

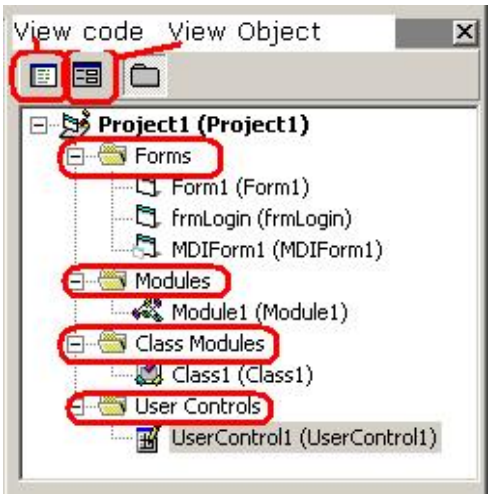


Figura 2.10. La ventana de proyecto

La ventana de propiedades

Todos los objetos tienen un conjunto de propiedades que lo describen, que definen su forma y comportamiento.

Algunas propiedades de los objetos son:

- Propiedad *Name*, que es el nombre con que se identifica el objeto y lo hace único ante los demás objetos.
- Propiedad *Caption*, contiene un texto como puede ser el título del formulario, el contenido de una etiqueta.
- Propiedad *Text*, contiene una cadena que usualmente se captura o selecciona en una caja de texto.

Existen otras muchas más como *tamaño, posición*, *color*, si está *activo* o no (*Enabled*), etc.

La ventana propiedades muestra del lado izquierdo el nombre de la propiedad, y del lado derecho, los valores que la propiedad es capaz de recibir.

Figura 2.11. Ventana de propiedades

Todos los valores de las propiedades se almacenan dentro de cada control o formulario en forma de *estructura* dentro de los archivos frm. Existen propiedades que se pueden cambiar en tiempo de ejecución y algunas sólo en el diseño. Para cambiar el valor de una propiedad durante el diseño del programa, se debe seleccionar al objeto y cambiar el valor en la ventana de propiedades. Para muchas de las propiedades los valores se encuentran acotados, lo que permite que *Visual Basic* muestre la lista de valores permitidos, ayudando al programador que los desconoce.

2.2. Creación de una aplicación

Construir aplicaciones con *Visual Basic* es sencillo, basta arrastrar los controles de la caja de herramientas al formulario, establecer sus *propiedades* en la ventana de propiedades y programar las acciones adecuadas en respuesta a los *eventos* o acciones que se dan, compilar la aplicación y crear el programa de instalación para tener una aplicación distribuible.

Para crear una aplicación debemos:

1. Agregar una forma.
2. Agregar los controles necesarios.
3. Definir el valor de las propiedades de los controles.
4. Si es necesario, crear la conexión a la base de datos.
5. Si es necesario, programar el comportamiento de los controles.
6. Compilar y depurar el programa.
7. Crear el archivo ejecutable.

Una vez probada la aplicación podemos crear el archivo ejecutable para su distribución. Para crear el archivo ejecutable seleccionamos la opción *Make nombreProyecto.exe…* en el menú *File,* generando un archivo con extensión *.exe*.

Para que el programa pueda correr es suficiente que el archivo *MSVBVM60.DLL* esté instalado en el directorio *c:\Windows\System* o *c:\Winnt\System32*. En el caso de proyectos más complejos en los que se utilicen muchos controles pueden ser necesarios más ficheros, la mayoría de ellos con extensiones *ocx*, *vbx* o *dll*.
Este problema se resuelve utilizando la utilería de creación de discos o archivos de instalación que forma parte de la suit de *Visual Basic y* se trata en el último tema.

El editor de código

El *editor de código* o *Code Editor* de *Visual Basic* fundamentalmente es la ventana de un procesador de textos, que nos permite escribir secuencias de instrucciones en lenguaje basic.

Existen 3 formas de abrir la ventana de editor de código:

1. Elegir *Code* en el menú *View*
2. Seleccionar el botón *View Code* de la ventana *Project Window*
3. Dando dos click en el formulario o en alguno de sus controles.

Figura 2.12. Ventana de código

La ventana de edición utiliza *código de colores* (accesible y modificable en *Tools/Options/Editor Format*) para destacar cada elemento del programa. Esta ayuda visual permite detectar y corregir problemas con más facilidad.

- Negro, el código escrito por el usuario.
- Azul, las palabras clave de *Basic*
- Verde, los comentarios
- Rojo, los errores.

En la parte superior de la ventana aparecen dos *listas desplegables*. La de la izquierda corresponde a los distintos elementos del formulario. La lista desplegable de la derecha muestra los distintos procedimientos que se corresponden con el elemento seleccionado en la lista de la izquierda.

El depurador de código

Una parte del tiempo de elaboración de un programa se destina a la *detección y corrección de errores*. La herramienta utilizada para ello es el *Depurador* o *Debugger*. La característica principal del *Debugger* es que permite ejecutar parcialmente el programa, deteniendo la ejecución en alguna(s) línea(s) y revisar el valor de las variables. De esta manera se facilita la detección de las fuentes de errores.

El lenguaje de programación Visual Basic

Comentarios

Todo lo que está a la derecha del *símbolo (')* es un *comentario*, por ejemplo:

```
’ Esto es un comentario
A = B*x+3.4 ' también esto es un comentario
```

Ámbito de las variables y los procedimientos

"Se entiende por *ámbito* de una variable la parte de la aplicación donde la variable es *visible* (accesible) y puede ser utilizada en cualquier expresión". (García de Jalón, 1999: 31)

Variables y funciones de ámbito local y global

Las variables, procedimientos y funciones, pueden comportarse como públicos (globales) o privados (locales), dependiendo el lugar donde se realizó la declaración y la palabra utilizada.

- Públicos

Las variables, procedimientos y funciones declaradas dentro de un módulo (archivo *.bas) con el prefijo *Public*, se convierten en públicas, y se puede acceder libremente desde cualquier punto del proyecto, ejemplo:

```
Public Sub Procedimiento1 (Parametro1 As Integer, …)
Public Function Funcion1 (Parametro1 As Integer,…) As Integer
Public var1_global As Double, var2_global As String
```

Para utilizar una variable *Public* o llamar a una función *Public* definidas en un formulario desde otro módulo se debe preceder el nombre de la variable o procedimiento con el nombre del formulario al que pertenece, ejemplo:

```
Módulo1.Variable1
Call Módulo1.Procedimiento1(Parametro1, …)
Retorno = Módulo1.Funcion1(Parametro1, …)
```

- *Local*

*S*i es definida dentro de un procedimiento o función. Las variables locales no son accesibles más que en el procedimiento o función en que están definidas. Para hacer que el valor de la variable se conserve hay que declarar la variable como *static* (como por ejemplo: *Static n As Integer*).

Una variable *Private*, por el contrario, no es accesible desde ningún otro módulo distinto de aquel en el que se haya declarado.

La siguiente tabla muestra la accesibilidad de las variables en función de dónde y cómo son declaradas.

Tipo de Variable	Lugar de declaración	Accesibilidad	Ámbito
Global o public	Declaraciones *.bas	Desde todos los formularios	Pública
Dim o private	Declaraciones *.bas	Desde todas las funciones de ese módulo	Local
Public	Declaraciones *.frm	De cualquier procedimiento del propio formulario y desde otros precedida por el nombre del módulo en que está declarada	Pública
Dim o private	Declaraciones *.frm	Desde cualquier procedimiento del propio formulario	Local
Dim	Cualquier procedimiento de módulo	Desde el propio procedimiento	Local

Cuadro 2.1. Accesibilidad de las variables.

Identificadores

“Un identificador es un nombre simbólico que se refiere a un dato o programa determinado” (García de Jalón, 1999: 33). Una variable es un identificador.

Nombres de variables

El nombre de una variable (o de una constante) tiene que cumplir las siguientes reglas:

- Comenzar siempre por una letra.
- Puede tener una longitud de 255 caracteres.
- No se admiten espacios o caracteres en blanco, ni puntos (.) ni otros caracteres especiales.
- Los caracteres pueden ser letras, dígitos, el carácter de subrayado (_) y los caracteres de declaración del tipo de la variable (%, &, #, !, @, y $).
- El nombre de una variable no puede ser una *palabra reservada* del lenguaje (*For*, *If*, *Loop*, *Next*, *Val*, *Hide*, *Caption*, *And*, ...).
- *Visual Basic* no distingue entre minúsculas y mayúsculas.

Constantes

Las *Constantes* son identificadores pero con la particularidad de que el valor que tienen asociado en la memoria sólo puede ser asignado una vez, es decir, el valor no cambia y permanece a lo largo de la ejecución de un programa. Para declarar un dato como constante es necesario utilizar la palabra reservada *Const* en la declaración. Ejemplo:

```
Const pi = 3.141516 ’ Las constantes son privadas por defecto.
Public Const MyString = "hola" ' Declaración de una constante pública.
Private Const MyInt As Integer = 5 ' Declaración de un entero constante.
Const Str = "Hola mundo", PI As Double = 3.14 ' Múltiples constantes en una línea.
```

Tipos de datos

Como todo lenguaje de programación *Visual Basic* cuenta con sus tipos de datos primitivos

Tipo	Descripción	Carácter de declaración	Rango
Boolean	Binario		True o False
Byte	Entero corto		0 a 255
Integer	Entero (2 bytes)	%	-32768 a 32767
Long	Entero largo (4 bytes)	&	-2147483648 a 2147483647
Single	Real simple precisión (4 bytes)	!	-3.40E+38 a 3.40E+38
Double	Real doble precisión (8 bytes)	#	-1.79D+308 a 1.79D+308
Currency	Número con punto decimal fijo (8 bytes)	@	-9.22E+14 a 9.22E+14
String	Cadena de caracteres (4 bytes + 1 byte/car hasta 64 K)		$ 0 a 65500 caracteres.
Date	Fecha (8 bytes)		1 de enero de 100 a 31 de diciembre de 9999. Indica también la hora, desde 0:00:00 a 23:59:59.
Variant	Fecha/hora; números enteros, reales, o caracteres (16 bytes + 1 byte/car. en cadenas de caracteres)	ninguno	F/h: como Date números: mismo rango que el tipo de valor almacenado
User-defined	Cualquier tipo de dato o estructura de datos. Se crean utilizando la sentencia Type		

En *Visual Basic* no es estrictamente necesario declarar todas las variables antes de utilizarlas a no ser que se elija la opción *Option Explicit* que hace obligatorio declararlas.

Operadores

El conjunto de operadores que soporta *Visual Basic* es:

Tipo	Operación	Operador
Aritmético	Potencia	^
	Multiplicación, división	*, /
	División entera	\
	Residuo	Mod
	Suma y resta	+,-
Cadena	Concatenación	&, +
Comparación	Igual a	=
	Menor que, menor o igual	<, <=
	Diferente	<>
	Mayor que, mayor o igual	>=
Lógicos	Negación	Not
	Y	And
	O inclusivo	Or
	O exclusivo	Xor

Estructuras de control

Se trata de estructuras importantes que permiten controlar el *flujo* de un programa, se dividen en:

- *Selección.* Permiten tomar decisiones.
- *Repetición*. Realizan un proceso repetidas veces..

Estructuras de selección:

- IF THEN ELSE

Esta estructura ejecuta el bloque A, si la condición es verdadera o el bloque B si es falsa. El bloque B es opcional. La condición es una expresión que al ser evaluada regresa el valor de cierto o falso.

```
If condicion Then
        Bloque A
[ Else
        Bloque B ]
End If
```

- Select Case

Esta sentencia permite ejecutar una de entre varias acciones en función del valor de una expresión.

Su forma general es la siguiente:

```
Select Case expresion
        Case etiq1
                [Bloque1]
        Case etiq2
                [Bloque2]
        Case Else
                Bloque N
End Select
```

Donde:

- *expresion* es una expresión numérica o alfanumérica,
- *etiq1*, *etiq2*,... pueden una *expresión.*

Estructuras de repetición

- For ... Next

La estructura *For* permite ejecutar un bloque de instrucciones un numero N de veces. Su forma general es:

```
For variable = expresion1 To expresion2 [Step expresion3]
        [Bloque]
        Exit For
        [Bloque]
Next [variable]
```

Cuando se ejecuta un *For*, se asigna el valor de *expresion1* a *variable* y se comprueba si su valor es mayor o menor a *expresion2.* En caso de ser menor se ejecuta el bloque de instrucciones, en caso de ser mayor el flujo de control sale del *for* y pasa la línea que está a continuación de *Next*. Esto sucede en caso de ser la *expresion3* positiva. En caso contrario se ejecutarán las instrucciones cuando variable sea mayor que *expresion2.* Una vez ejecutadas las instrucciones, la variable se incrementa en el valor de la *expresion3*, o en 1 si *Step* no se especifica, volviéndose a efectuar la comparación entre variable y la *expresion2*, y así sucesivamente.
La sentencia *Exit For* es opcional y permite salir de un bucle *For... Next* antes de que éste finalice.

- Do ... Loop

Esta estructura permite repetir la ejecución de bloque de instrucciones mientras una condición dada sea verdadera. La condición puede ser verificada antes o después de ejecutarse el bloque de instrucciones. Sus posibles formas son las siguientes:

```
Forma1:
    Do [{While/Until} condicion]
        [BloqueA]
    [Exit Do]
        [BloqueB]
    Loop

Formato 2:
    Do
        [BloqueA]
    [Exit Do]
        [BloqueB]
    Loop [{While/Until}condicion]
```

La sentencia opcional *Exit Do* permite salir de un bucle *Do... Loop* antes de que finalice.

- While … Wend

Esta estructura es otra forma de implementar ciclos de repetición mientras se cumpla la condición. Su estructura es la siguiente:

```
While condicion
    [BloqueA]
Wend
```

Funciones y procedimientos

Un *procedimiento* es un subprograma, que al ser llamado no regresa valores. A diferencia de una *función* que siempre regresa un valor.

Funciones (function)

La sintaxis correspondiente a una función es la siguiente:

```
[Static] [Private] Function nombre ([lista de parámetros]) [As tipo]
        [
        [nombre = expresion]
        [Exit Function]
        [
End Function
```

Donde:

- *nombre* es el nombre de la función. Será de un tipo u otro dependiendo del dato que devuelva.
- *Lista de parámetros* son los argumentos que son pasados cuando se llama a la función.
- *nombre* de la función actúa también como una variable dentro del cuerpo de la función. El valor de la variable *expresion* es almacenado en el propio nombre de la función.
- *Exit Function* permite salir de una función antes de que ésta finalice y devolver así el control del programa a la sentencia inmediatamente a continuación de la que efectuó la llamada a la función.
- *End Function* marca el final del código de la función y, al igual que la *Exit Function*, devuelve el control del programa a la sentencia siguiente a la que efectuó la llamada.

Procedimientos (sub)

Su sintaxis es:

```
[Static] [Private] Sub nombre [(lista de parámetros)]
        [Bloque A]
        [Exit Sub]
        [Bloque B]
End Sub
```

La llamada a un *procedimiento* puede ser de alguna de las dos formas siguientes:

- Call nombre[(lista de parámetros)]

  Utilizando la palabra Call y colocando los parámetros entre paréntesis.

- nombre [lista de argumentos]

  Utillizando el nombre del procedimiento y a continuación los argumentos separados por comas.

Un *procedimiento* no puede ser utilizado en una expresión pues no devuelve ningún valor. Una función puede ser llamada como un *procedimiento*, en este caso no se hace nada con el valor devuelto por la función.

Paso de parámetros por referencia y por valor

En las *funciones* y en los *procedimientos*, el paso de los parámetros puede dividirse en dos:

- Por *referencia*

  Es el paso de parámetros por default en Visual Basic donde cualquier cambio de valor que sufra un parámetro dentro de la *función* o del *procedimiento* también se produce en el argumento correspondiente debido a que se está trabajando con la variable original.

- Por valor

  Cuando se llama a una *función* o a un *procedimiento*, se puede especificar que el valor de un argumento *no sea cambiado*, poniendo al argumento entre paréntesis en la llamada. En el paso de parámetros por valor, se crea una nueva variable a la que se le copia el valor de la variable original, de tal forma que la variable original no sufre cambios pues se trabaja con la copia y ésta es la que se manipula.

Explícitamente se puede declarar dentro de la función o procedimiento si el paso de parámetros es por valor utilizando la palabra reservada *byVal* o *byRef* para un paso de parámetros por referencia.

Bibliografía del tema 2

Ceballos, Francisco Javier, (2004), *Curso de programación de Visual Basic 6,* 2ª Edición, México, Alfaomega-Rama.

Perry Greg y Hettihewa Sanjaya, (2004), *Aprendiendo Visual Basic 6 en 24 horas*, México, Prentice Hall.

García de Jalón Javier, (1999), *Aprenda Visual Basic 6.0 como si estuviera en primero*, San Sebastián.

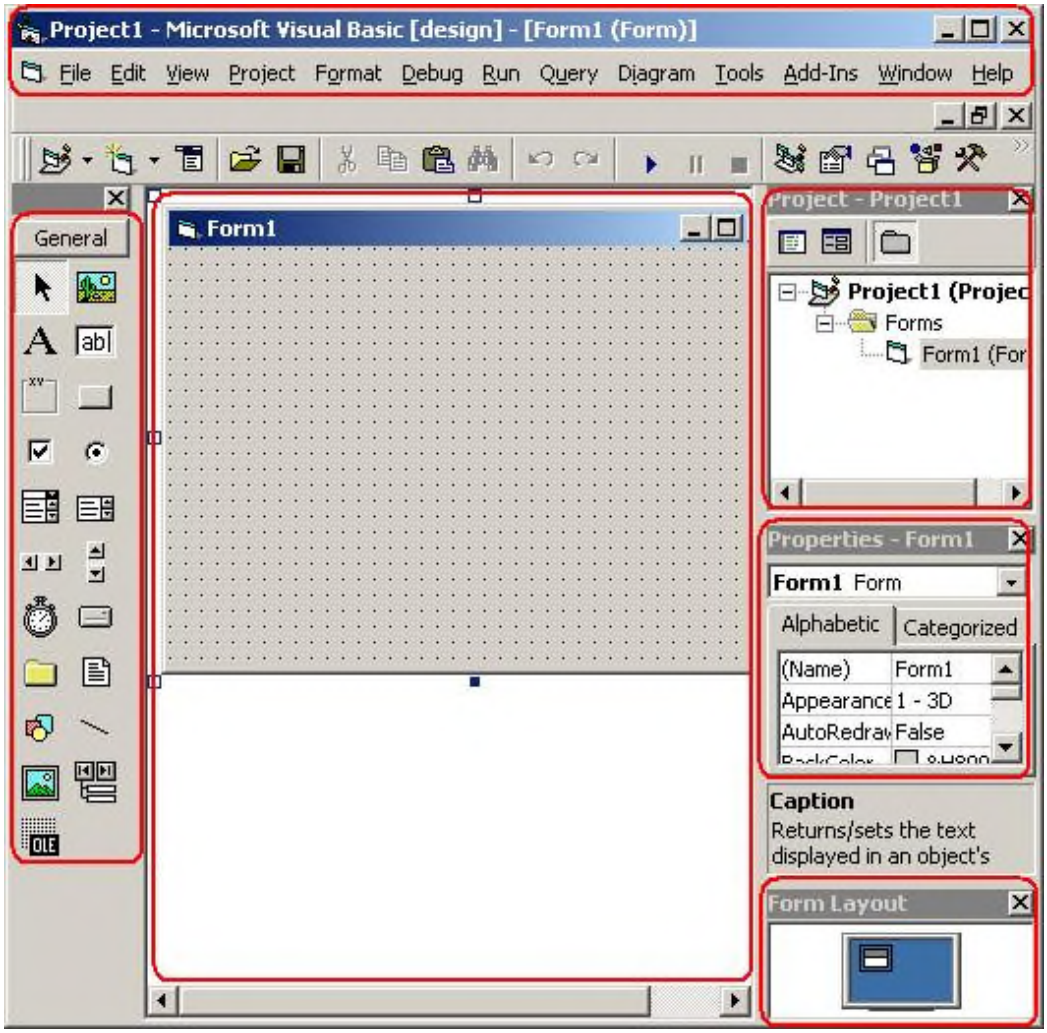
Actividades de aprendizaje

A.2.1. Revisa el entorno de programación propuesto con Visual Basic 6.0 e indica las ventajas con respecto al uso de un entorno de desarrollo con un lenguaje de 3GL.

A.2.2. Completa la siguiente tabla indicando la accesibilidad de las variables en función de dónde y cómo son declaradas.

Tipo de variable	Lugar de la declaración	Accesibilidad
Global		
Public		
Dim		
Private		

A.2.3. De la siguiente imagen, nombra cada una de las partes que forman parte del entorno de desarrollo de Visual Basic.



Cuestionario de autoevaluación

1. ¿Qué entiendes por lenguaje de cuarta generación?
2. ¿Cuándo debemos utilizar un lenguaje de cuarta generación y cuándo no?
3. ¿Qué es una DLL?
4. ¿Cuáles son los elementos del entorno de desarrollo de Visual Basic?
5. ¿Qué es un punto de ruptura?
6. ¿Cuál es la diferencia entre una función y un procedimiento?
7. ¿Cuál es la diferencia entre una variable global y una local?
8. ¿Cuándo debemos utilizar un *For... next, do... loop y while… wend*?
9. ¿Cuál es la diferencia entre un paso de parámetros por valor y por referencia?
10. ¿Qué son los tipos de datos primitivos de un lenguaje de programación?

Examen de autoevaluación

Elige la opción correcta

1.- Los 4gl comprenden:

a) Lenguajes de presentación, lenguajes especializados, generadores de aplicaciones, lenguajes de muy alto nivel

b) Lenguajes de propósito general, de presentación, especializados, generadores de aplicaciones

c) Lenguajes de presentación, de bajo nivel, orientados a objetos

d) Lenguajes regulares, de presentación, representados por gramáticas de contexto libre

e) Ninguna de las anteriores

2. Son aplicaciones que podemos construir en el entorno de Visual Basic:

a) Programas estándar (.com y exe), componentes active X, librerías DDL

b) Programas estándar (.exe), controles active X, bibliotecas dll, aplicaciones web

c) Programas estándar (.com), componentes active X, bibliotecas DDL

d) Programas de procesamiento por lotes, programas estándar (.com y exe), programas en bytecode

e) Ninguna de las anteriores

3.- ¿Son elementos del entorno de desarrollo de Visual Basic?

a) Barra de menú y de herramientas, caja de herramientas, zona de formularios, ventana de proyecto, la ventana de forma

b) Barra de título, de menú, de herramientas y de componentes, caja de herramientas, zona de formularios, ventana de proyecto, la ventana de forma

c) Barra de título, de menú y herramientas, caja de herramientas, zona de formularios, ventana de proyecto, la ventana de forma, ventana de controles

d) Barra de título, de menú y herramientas, caja de herramientas, zona de formularios, ventana de proyecto, la ventana de forma, ventana de propiedades

e) Barra de título, de menú y herramientas, caja de herramientas, zona de formularios, ventana de proyecto, ventana de propiedades

4.- ¿Las variables declaradas con la palabra (dim x as integer) o private (private x as integer) declaradas en el módulo (archivo *.bas) se pueden ver desde?

a) Todos los formularios, desde todos los módulos

b) Todas las funciones del módulo donde están declaradas

c) Desde cualquier procedimiento del formulario

d) Son locales al formulario

e) Ninguna de las anteriores

5.- ¿Las variables DIM declaradas dentro de cualquier procedimiento se pueden ver desde?

a) Todos los formularios, desde todos los módulos
b) Todas las funciones del módulo donde están declaradas
c) Desde cualquier procedimiento del formulario
d) Son locales al formulario
e) Solo dentro del propio procedimiento

6.- Son tipos de datos primitivos de Visual Basic

a) Boolean, byte, integer, long, single, double, string, date, variant, float
b) Boolean, byte, integer, long, single, double, string, date, variant
c) Boolean, byte, int, long, single, double, string, date, variant, currency
d) Boolean, byte, integer, long, single, double, string, date, real
e) Float, byte, integer, long, single, double, string, date, variant

7.- Son operadores soportados por Visual Basic

a) +, *, /, mod, \, =, <=, >=, !=, not, and, or, xor
b) +, *, /, mod, \, =, <=, >=, <>, not, and, or, xor
c) +, *, /, mod, \, =, =>, >=, !=, not, and, or
d) +, *, /, mod, =, <=, =>, <>, not, and, or, xor, div
e) +, *, /, mod, =, <=, >=, !=, not, and, or, xor, div

8.- Son estructuras de selección de Visual Basic

a) If…then..else…endif, Select case
b) If…then..else…endif, Select switch
c) If…then..else…endif, Select case, do..loop
d) If…then..else…endif, switch case
e) Select case, iff, If…then..else…endif

9.- Son estructuras de repetición

a) For.. until, Do..loop, While…wend

b) For…Next, Do…loop, while…wend

c) Switch case, for…next, do…loop

d) Repeat…until, for…next, while…until

e) Repeat…until, do…loop, while…wend

10.- Para indicar que el paso de parámetros es por valor se utiliza la palabra reservada:

a) ByRef

b) ByVal

c) No hay paso de parámetros por valor, todos son por referencia

d) &<nombre variable>

e) Es indistinto, siempre el paso de parámetros es por valor

11.- Para indicar que el paso de parámetros es por referencia se utiliza la palabra reservada

a) ByRef

b) ByVal

c) No hay paso de parámetros por referencia, todos son por valor

d) *<nombre variable>

e) Es indistinto, siempre el paso de parámetros es por referencia

TEMA 3. HERRAMIENTAS DE ACCESO A UNA BASE DE DATOS

Objetivo particular

El alumno reconocerá el proceso para la creación de aplicaciones basadas en Visual Basic a partir de conexiones de bases de datos, identificando las ventajas y desventajas entre ellas, y comparando la forma de hacerlo con un lenguaje de tercera generación.

Temario detallado

3.1 Ole DB

3.2 Usos de Active-X Data Object

Introducción

“Una base de datos puede definirse como un conjunto ordenado de datos relacionados entre sí, almacenados, estructurados, no redundantes y de fácil acceso”. (Rob, 2003: 7)

Una base de datos se parece a un archivero electrónico bien organizado en el que un software, conocido como Sistema de Administración de Bases de Datos (DBMS por sus siglas en inglés) ayuda a manejar la estructura de la base de datos y controla el acceso a los datos guardados en la base de datos. El DMBS también permite compartir los datos de la base de datos entre múltiples aplicaciones y usuarios.

Desde que aparecen los sistemas manejadores de bases de datos se tiene una nueva forma de almacenar los datos, ya que estos implementan en una

arquitectura todas las herramientas para la creación y ejecución de consultas de forma optimizada.

Una de estas arquitecturas para organizar la información es el modelo de Bases de Datos Relacional, en el que los datos son organizados en conjuntos lógicos y matemáticos en una estructura tabular, donde cada campo representa una columna de la tabla y cada registro un renglón. Es decir, cada tabla es una matriz compuesta por una serie de intersecciones de filas y columnas. Las tablas, también llamadas relaciones, están relacionadas entre sí porque comparten una característica de entidad común.

Hoy en día, la tarea del manejador de base de datos es planear la organización y acceso a la base de datos y sólo requerimos contar con un motor de base de datos como lo es JetDb de Microsoft o tener una conexión a una base datos para poder manipular la información.

3.1 Ole DB

OLE DB es un conjunto de interfaces COM de programación que permiten acceder a bases de datos locales o pequeñas (Microsoft Access, Btrieve, dBase, FoxPro, Microsoft Excel, archivos de texto ASCII, etc.), bases de datos remotas o corporativas (Microsoft SQL Server, Oracle, Informix, etc.). Así como a cualquier origen de datos que pueda representarse en formato de filas y columnas. A nivel de sistema para acceder a cualquier tipo de datos, manteniendo un bajo uso de memoria y espacio en disco.

OLE DB proporciona a las aplicaciones un procedimiento unificado de acceso a los datos almacenados en diferentes recipientes de información, generalmente bases de datos relacionales o no relacionales. Permite utilizar las numerosas

prestaciones de DBMS (Data Base Management System) apropiadas para el origen de datos y permite compartir sus datos.

Conceptualmente, este estándar utiliza tres tipos de componentes:

- Data Provider: Son aplicaciones como SQL Server o sistemas operadores de componentes, como archivos de sistema, que permiten acceder a datos.
- Data Consumer: Son aplicaciones que usan los datos disponibles por un Data Provider.
- Service Component: Son componentes que procesan y transportan datos.

VB incluye herramientas visuales que requieren poco o nada de código para interactuar con un Data Source (origen de datos) como el Data Environment Designer y el ADO Control.

- El Data Environment Designer permite crear objetos que accedan a datos. Se utiliza para crear conexiones a bases de datos y comandos para recuperar y manipular datos.
- El ADO control es un control gráfico usa Active X Data Objects (ADO) para crear conexiones rápidas entre controles ligados a datos (data-bound controls) y un data source, permitiendo crear aplicaciones con un mínimo de código.

“ADO es el modo en que los programadores obtienen acceso a OLE DB. Todos los nuevos controles enlazados a datos, el entorno de datos y el Diseñador de informe de datos son compatibles con OLE DB.”[4]

En la presentación de datos, se agregan controles ligados a datos (data-bound) en un formulario y se une cada uno a los campos de un Data Source, la primera

[4] Luis Franco, Tutorial de ADO-1 Material en línea, disponible en: http://www.uyssoft.com p. 1 Fecha de recuperación: 9 de diciembre 2008.

forma de hacerlo es arrastrando y pegando los elementos de un Data Environment en un formulario, y la segunda es la forma manual utilizando las siguientes propiedades:

- *DataSource*, nombre del Data Environment.
- *DataMember*, nombre del objeto command.
- *DataField*, nombre del campo en un command.

3.2 Usos de Active-X Data Object

El Active-X Data Object es conocido como ADO y permite acceder y manipular datos desde un origen de datos (usualmente una base de datos) concediendo crear aplicaciones escribiendo código para manejar propiedades, eventos y métodos. La tecnología ADO utiliza tres objetos principales:

- Objeto *Connection*. Se utiliza para crear una conexión entre la aplicación y un Data Source a través de un proveedor. Las principales propiedades que utiliza son: *Provider, ConnectionString*
- Objeto *Command*. Se utiliza para construir *queries (consultas)* para manipular datos dentro de un *Data Source* o llamando a un procedimientro almacenado de SQL Server.
- Objeto *Recorse*t. Se usa para guardar los registros que dan como resultado de una consulta (query) en SQL. Permite navegar, modificar, guardar o borrar registros específicos

Para hacer uso de los objetos ADO debemos agregar el componente Microsoft ADO data control al proyecto. Figura 3.1.

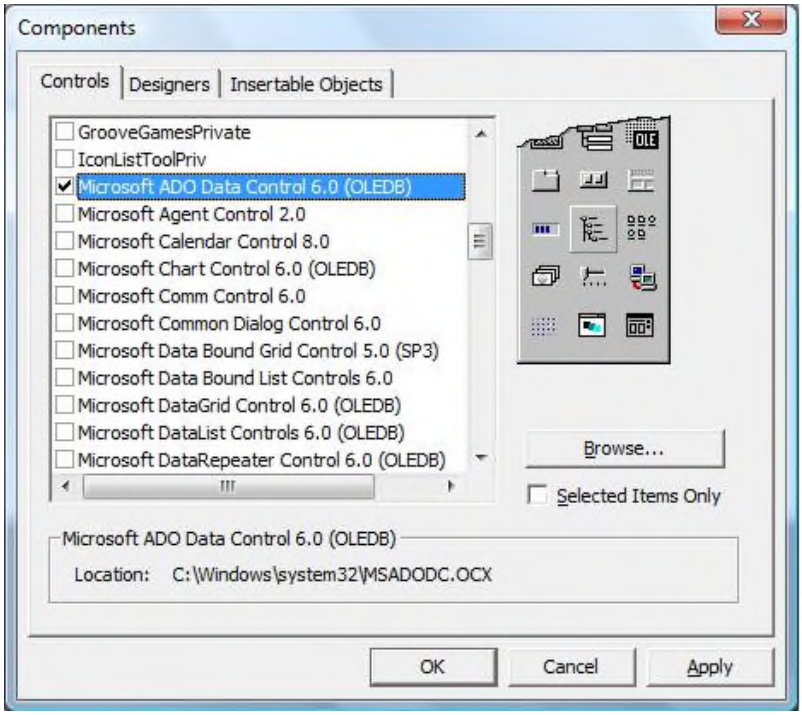


Figura 3.1. Agregar ADO al proyecto (proyect --> Components --> Controls)

El control de datos ADO (ADO Data Control, ADODC)

Este control permite de forma fácil establecer una conexión a la base de datos y manipular los datos desde un formulario, ya que sin programar ni una línea de código podemos agregar, modificar y eliminar información de la base de datos.

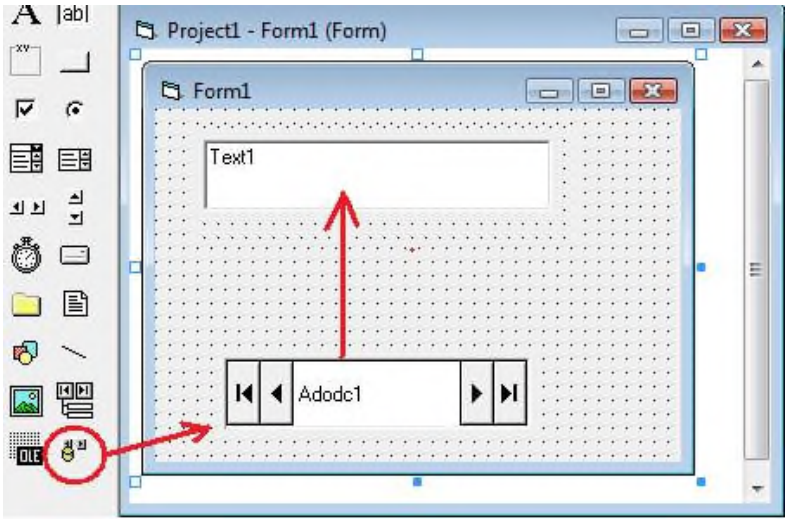


Figura 3.2. El control de datos ADO.

Se utilizan Objetos de Datos ActiveX (ADO) para crear una conexión al origen de datos y proveer de ellos a los controles de enlace de datos. Los controles de enlace de datos son aquellos que tiene la capacidad de enviar/recibir datos de un control de datos por medio de propiedad *DataSource*. En la figura 3.2, la caja de texto es un control de enlace de datos, ya que por medio de la propiedad *dataSource* se liga al origen de datos (adodc1), y por la propiedad *dataField* se liga a algún campo de la tabla.

El control de datos cuenta con los botones de primero, anterior, siguiente y último, que permiten navegar entre los registros de la tabla a la que se encuentra ligado el control. Al momento de navegar, el control de enlace muestra los datos del registro activo de la tabla.

La mayor parte de los controles de Visual Basic pueden ser de enlace de datos, como: CheckBox, ComboBox, Image, Label, ListBox, PictureBox, TextBox, DataGrid, DataCombo, Chart y DataList. También pueden crearse controles ActiveX de enlace de datos, o comprar controles de otros proveedores. Es importante tener presente en todo momento la compatibilidad entre los controles porque no son compatibles todos los controles. Podemos comenzar a desarrollar un proyecto con un conjunto de controles (por ejemplo: bajo DAO) y por funcionalidad agregamos un control (ADO) que no es compatible. Entonces debemos de decidir entre permanecer bajo la misma tecnología o migrar a la nueva.

Acceso a datos con ADO

Las aplicaciones con ADO tiene la siguiente arquitectura. Figura 3.3.

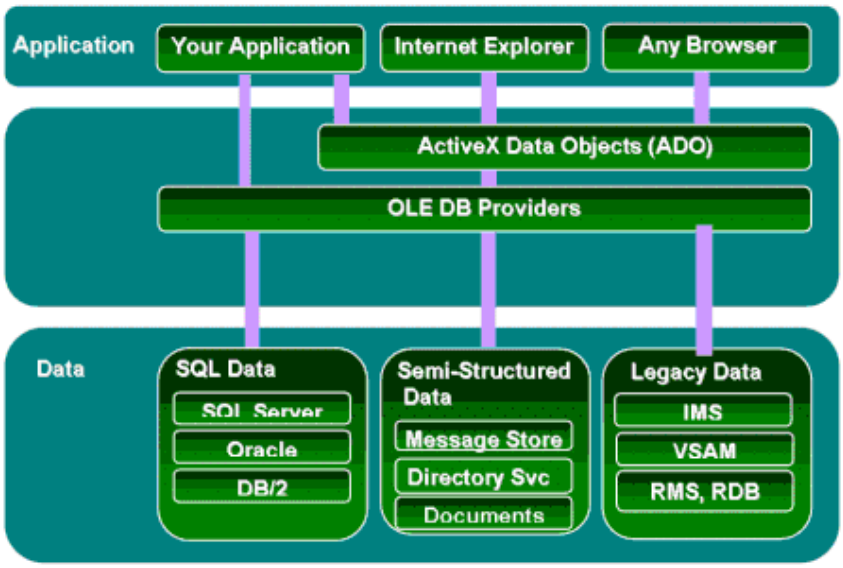


Figura 3.3. Arquitectura de aplicaciones ADO[5]

El control de origen de datos se coloca dentro del formulario, este por medio de ADO se conecta con el proveedor de datos (OLE DB) quien se encarga de acceder a los datos.

Lista de pasos para crear una aplicación con acceso a una base de datos

Básicamente podemos decir que son dos pasos los que debemos hacer para crear una aplicación con acceso a una base de datos.

Paso 1: Crear una conexión a una base de datos

Lo primero que debemos hacer es configurar al control *ADO data control (Adodc)* para que pueda conectarse con la base de datos, por medio de su propiedad *ConecctionString*. Que puede contener:

- Un archivo de vínculo de datos OLEDB (*.UDL).
- Un nombre de origen de datos de ODBC (DSN).

[5] Imagen tomada del MSD1.0

- Una cadena de conexión.

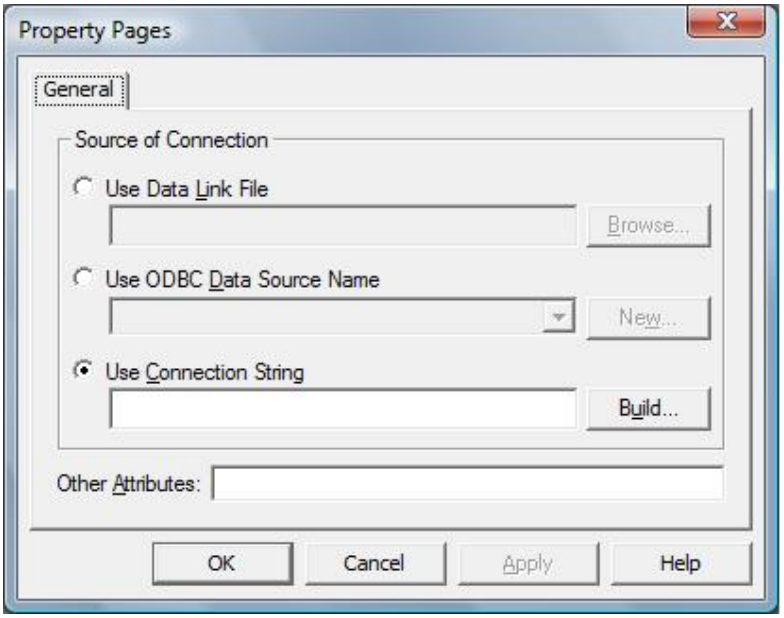


Figura 3.4. Ventana para seleccionar el tipo de conexión

En esta parte se debe considerar cuál de las 3 formas se debe utilizar:

- ODBC. Solo en casos extremos, cuando OLE DB no pueda conectarse directamente a la base datos y sea necesario utilizar el proveedor de datos abierto. Una de las consecuencias es que el ODBC se debe configurar en cada una de las máquinas donde corre la aplicación. Esta configuración se debe hacer de forma manual o por medio de un script.
- Data Link File. La cadena de conexión se guarda en un archivo UDL, permitiendo cambiarla editando al archivo. Esta forma nos permite hacer aplicaciones más robustas. Pero debemos tener más cuidado, ya que el archivo debe formar parte del paquete de instalación y su ubicación debe ser cuidadosa.
- Connection String. Es la forma más fácil de conectarse a la base de datos. Se realiza la conexión una sola vez y se almacena dentro de la forma, como un atributo de ADO data control. Al estar dentro de la aplicación garantizamos que siempre viajará con la aplicación. Presenta una

desventaja al momento de cambiarla, ya que se debe cambiar en el código fuente y compilar nuevamente la aplicación.

La recomendación es: Desarrollar la aplicación al inicio con *connectionstring* y a medida que madure el proyecto cambiar al *dataLinkFile*.

Cuando construimos la cadena de conexión se indica: *Nombre del usuario, password, ubicación de las bases de datos, proveedor o motor de conexión.* Estos parámetros se proporcionan en la ventana de *Data link properties.*

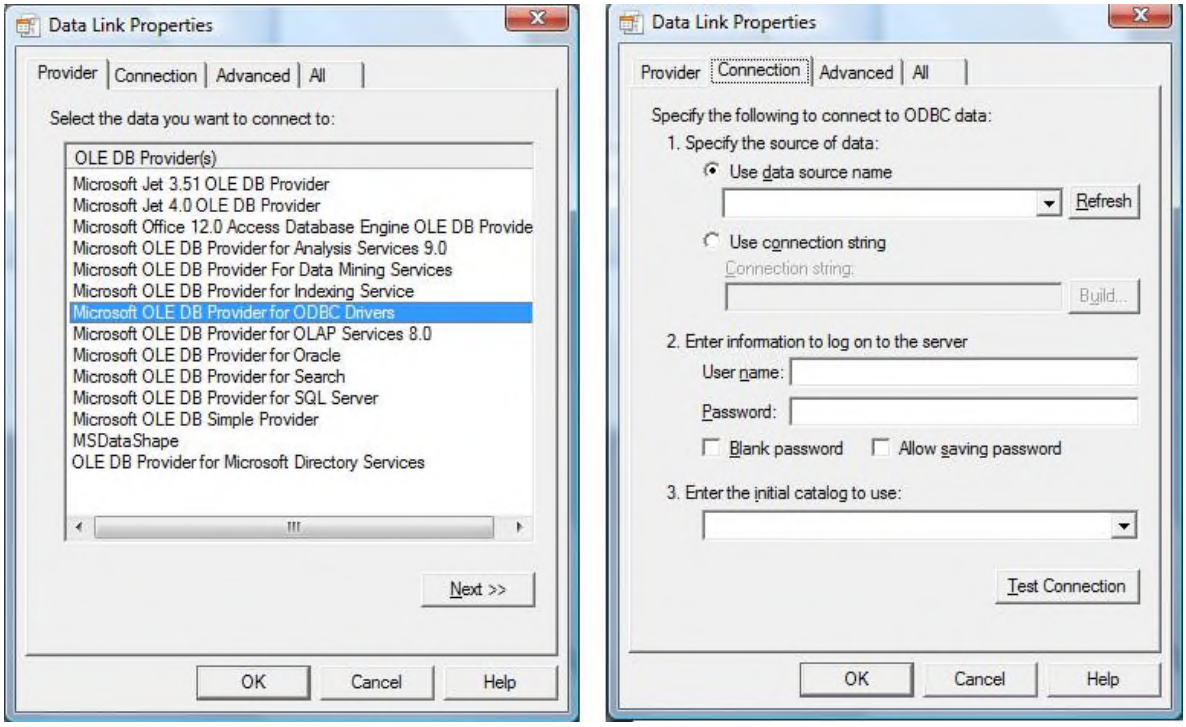


Figura 3.5. Ventana de *Propiedades de la Conexión* (Data Link Properties)

La ruta de acceso, el nombre de usuario y la contraseña se especifican en la segunda pestaña (conexión).

Paso 2: Vincular controles a un control de datos ADO

Una vez que el ADO Data Control se ha conectado con la base de datos y se ha definido su origen de datos, podemos utilizar los controles disponibles en la caja de herramientas para mostrar y editar la información contenida en el ADO Data Control. Cada control *TextBox* que utilicemos en el formulario para mostrar o editar información, debe ser vinculado al ADO Data Control a través de sus propiedades *DataSource* y *DataField*.

La propiedad *DataSource* define el origen de datos o a que ADO Data Control está vinculado este campo de texto.

La propiedad *DataField* determina el campo cuyo contenido se mostrará en el *TextBox*.

El control ADODC proporciona la funcionalidad de permitir desplazamiento por los registros del origen de datos seleccionado, así como permitir la modificación del registro actual sin tener que programar una línea de código.

Propiedades del Control de Datos ADO

Algunas propiedades del control de datos ADO son:

- *ConnectionString.*

  Cadena de conexión a la base de datos, que contiene los parámetros básicos que permiten al ADODC conectar con el proveedor OLE-DB.

- *RecordSource*

  Representa el nombre del origen de los registros dentro de la base de datos, el nombre de una tabla, una sentencia válida en ANSI SQL, el nombre de un procedimiento almacenado, etc.

- *Password y UserName*

  Definen respectivamente la contraseña y el nombre de usuario que se utiliza para conectase con la base de datos para ser verificados por el servidor. En

algunas situaciones, estos parámetros se escriben dentro de la cadena de conexión.

- BOF (*Begin of file)* y EOF *(End of file)*

El ADODC utiliza al objeto recordset para manipular los registros, este objeto cuenta con dos banderas para identificar si se encuentra al inicio o final de los registros, estas son BOF (*Begin of file)* y EOF *(End of file)*.

BOF indica si la posición del registro actual es anterior al primer registro de un Recordset. *EOF* indica si la posición del registro actual es posterior al último registro de un Recordset.

La siguiente tabla muestra los valores que pueden aceptar las propiedades *BOF* y *EOF*

Propiedad BOF/EOF del recordset	**Significado**
BOF y EOF ambos False	El RecordSet apunta en un registro "válido" (con datos).
BOF = Trae	El registro actual está situado antes del primer registro de datos. El registro actual no es "válido".
EOF = Trae	El registro actual está situado detrás del último registro de datos. El registro actual no es "válido".
BOF y EOF ambos Trae	No hay filas en el RecordSet. El registro actual no es válido. Es recordset está vacío.

Métodos del Control de Datos ADO

- *MoveFirst, MovePrevious, MoveNext, y MoveLast*

ADODC utiliza métodos para desplazarnos por los registros de un *RecordSet,* éstos métodos son *MoveFirst, MovePrevious, MoveNext, y MoveLast*, que

respectivamente de desplazan al primero, al anterior, al siguiente y al último de los registros.

La sintaxis requerida para desplazarse por los registros de un recordset ligado a un ADO control con es:

```
ADODC1.RecordSet.MoveFirst          'Se desplaza al primero
ADODC1.RecordSet.MovePrevious       'Se desplaza al anterior
ADODC1.RecordSet.MoveNext           'Se desplaza al siguiente
ADODC1.RecordSet.MoveLast           'Se desplaza al último
```

El método *Move* permite moverse dentro del recordset hacia atrás o adelante un número *n* de registros. Si el número es positivo se desplaza hacia delante (1 va al siguiente), y si es negativo, se desplaza hacia atrás (-1 va al anterior).

- *AddNew*

El método AddNew crea un registro en blanco para poder introducir nuevos valores.

- *Update*

Guarda los cambios realizados en el registro actual, si sale del registro que está modificando o agregando antes de llamar al método *Update*, ADO llamará automáticamente a *Update* para guardar los cambios. Debe llamar al método *CancelUpdate* si quiere cancelar los cambios hechos en el registro actual o desechar un registro agregado recientemente.

- *Delete.*

Elimina el registro actual o un grupo de registros.

Eventos del control de datos ADO

El control de datos dispone de un conjunto de eventos que se dan en el momento en que se cumplen determinadas acciones.

- WillMove / MoveComplete

El evento *WillMove* se da antes de que una operación modifique la posición actual del Recordset. El evento *MoveComplete* se da después de que la posición actual del *Recordset* haya cambiado.

- WillChangeField / FieldChangeComplete

El evento *WillChangeField* se da antes de que una operación pendiente modifique el valor de los campos del *Recordset*. El evento *FieldChangeComplete* se da después de que el valor de los campos haya sido modificado.

- *WillChangeRecord / RecordChangeComplete*

El evento *WillChangeRecord* se da antes de que uno o varios registros (filas) del *Recordset* se modifiquen. El evento *RecordChangeComplete* se da después de que uno o varios registros cambien.

- *WillChangeRecordset/RecordsetChangeComplete*

El evento *WillChangeRecordset* se da antes de que una operación pendiente modifique el *Recordset.* El evento *RecordsetChangeComplete* se da *después* de que el *Recordset* haya sido modificado.

- *Error*

Ocurre sólo como resultado de un error de acceso a los datos.

Bibliografía del tema 3

Ceballos, Francisco Javier, (2004), *Curso de programación de Visual Basic 6,* 2ª Edición, México, Alfaomega-Rama.

Vaughn, William R, (1999), *Programación SQL Server 7.0 con Visual Basic 6.0*, México, McGraw Hill.

Franco, Luis, *Tutorial de ADO-1*, Material en línea, disponible en: http://www.uyssoft.com. Fecha de recuperación: 9 de diciembre 2008.

Rob Peter, (2003), *Sistemas de Bases de Datos*, México, Thomson.

Actividades de aprendizaje

A.3.1. Completa la tabla con la información que se solicita acerca de las propiedades, eventos y métodos del ADO Control.

Propiedades		Eventos		Métodos	
Nombre	Descripción	Nombre	Descripción	Nombre	Descripción

A.3.2. Completa la tabla con la información que se solicita acerca de OLE DB y ActiveX Data Object.

OLEDB		ActiveX DataObject	
Componente	Descripción	Objeto	Descripción

Cuestionario de autoevaluación

1. ¿Qué es una base de datos?
2. Explica brevemente qué es una base de datos relacional
3. ¿Qué es OLE DB?
4. ¿Cuáles son los tres componentes del estándar OLE DB?
5. ¿Qué entiendes por conexión a una base de datos?
6. ¿Qué es ADO?
7. ¿Qué entiendes por recordset?
8. ¿Qué propiedades del objeto *TextBox* se utilizan para vincular con el objeto ADO Data Control?
9. ¿Podemos hacer programas de acceso a bases de datos prácticamente sin escribir líneas de código en Visual Basic, SI/NO?, ¿Cuáles son sus ventajas y desventajas?
10. ¿Qué es una conexión por ODBC?

Examen de autoevaluación

Elige la opción correcta

1.- ¿El Data Environment Designer permite?

a) Crear objetos que acceden a datos. Se utiliza para crear conexiones a bases de datos y comandos para recuperar y manipular datos.
b) Diseñar formas para acceder a bases de datos
c) Diseñar controles active X que se conectan a bases de datos por ODBC
d) Ni pertenece al entorno de desarrollo de Visual Basic
e) Ninguna de las anteriores

2.- ¿Son los objetos principales que utiliza ADO)?

a) Connection, recordsource, txt

b) Command, TextBox, data source

c) Connection, Recordset, dataMember

d) Connection, recordsetType, Command

e) Connection, Command, Recordset

3.- ¿Son propiedades de la caja de texto?

a) DataField, dataMember, dataSource, recordSource

b) DataField, dataMember, dataSource

c) DataField, dataMember, dataSource, readOnly

d) DataField, dataMember, dataSource, connectionSource

e) DataField, dataMember, dataSource, CursorLocation

4.- ¿Qué opciones puede contener la propiedad *ConnectionString* del objeto ADO Control al conectarse a una base de datos?

a) Cadena de conexi;on, un SQL, el nombre de base de datos

b) OLEDB, ODBC, el nombre de la base de datos

c) OLEDB, ODBC, Cadena de conexión

d) ODBC, dataSource, conection

e) Conecction, ODBC, OLEDB

5.- ¿Una cadena de conexión se puede almacenar?

a) Solo dentro de la aplicación

b) Solo en un archivo UDL

c) En la aplicación y en un archivo UDL siempre en ambos lados

d) En la aplicación o en un archivo UDL

e) No se almacena

6.- Son propiedades del DATA CONTROL (ADO)?

a) ConnectionString, cursorLocation, cursorType, lockType, recordSorce

b) DatabaseName, readOnly, recordsetType, recordsource, data source

c) DatabaseName, readOnly, recordsetType, recordsource, dataMember

d) DatabaseName, readOnly, recordset type, recordsource, cursorType

e) DatabaseName, readOnly, recordsetType, recordsource, connectionString, cursorLocation

7.- Si las propiedades del recordset BOF y EOF están en falso ¿qué significa?

a) El recordset no tiene datos

b) El recordset está lleno

c) El recorset apunta a un registro válido

d) El recorset no apunta a un registro válido

e) El recorset apunta al último registro

8.- Son métodos del AdoControl (ADO) para desplazarse entre registros

a) moveNext, movePrevious, moveLast, moveFirst

b) moveDown, moveUp, move left, moveRigth

c) gotoEnd, gotoFirst, gotoNext, gotoPrevious

d) moveNext, movePrevious, moveUp, move left

e) gotoEnd, gotoFirst, gotoNext moveDown

9.- Son métodos del AdoControl para modificar registros

a) update, delete

b) transform, update, delete

c) update, delete, erase

d) update, union, delete

e) ninguna de las anteriores

10.- Son eventos del control de datos ADO

a) afterUpdate, beforeUpdate, move

b) willMove, moveComplete, willChangeField, FieldChangeComplete

c) moveNext, movePrevious, moveFirts, moveLast

d) afterUpdate, beforeUpdate, willMove, moveComplete, willChangeField, FieldChangeComplete

e) Ninguna de las anteriores

TEMA 4. MÓDULOS DE CLASE DE UN LENGUAJE DE CUARTA GENERACIÓN

Objetivo particular

El alumno reconocerá los beneficios de utilizar los módulos de clase en la creación de componentes y clases con acceso a bases de datos.

Temario detallado

4.1. Introducción a clases

4.2 Data Bound Class Module

Introducción

“La Programación Orientada a Objetos (POO) es un nuevo paradigma en programación que considera a un programa como una colección de agentes considerablemente autónomos, llamados objetos. Cada objeto es responsable de tareas específicas” (Budd, 1994: 15-16). Esta tecnología utiliza clases para encapsular (es decir, envolver) datos (atributos) y métodos (algoritmo o conjunto de operaciones).

4.1 Introducción a clases

Una clase es una definición formal de un tipo de objeto, determina qué datos formarán parte del objeto (atributos), qué tareas desarrollará el objeto (comportamiento) y de qué manera va a interactuar con el usuario y con otros objetos.

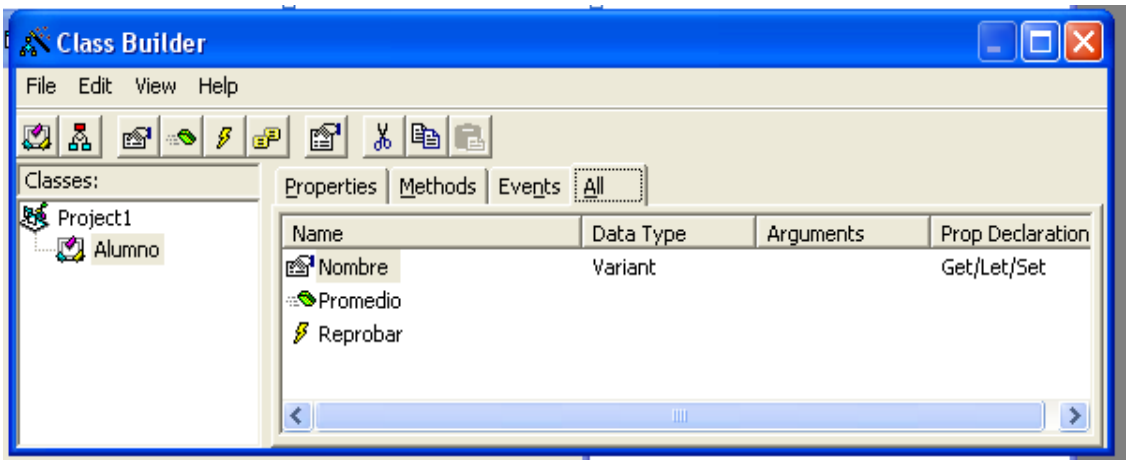
Al generar un objeto a partir de una clase realizamos una *instancia* de la clase, es decir, estamos haciendo una "copia funcional" de la clase y esta copia (objeto) "hereda" los atributos y métodos de la clase. Las clases y por lo tanto los objetos (instanciación) que se derivan de ellas son un conjunto de datos y comportamientos.

La forma de crear clases en un lenguaje de 4GL es haciendo uso del módulo de clase. El *módulo de clases* es un tipo de editor de código que va a permitir construir instrucciones que van a dar funcionalidad a un objeto, el cual será utilizado dentro de una aplicación.

Existen dos formas de crear una clase en Visual Basic (VB):

1. Generador de clases

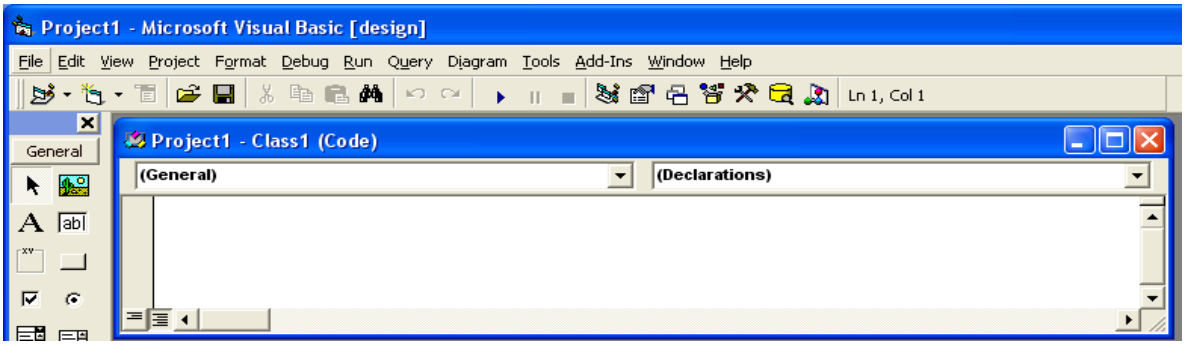
   Es la herramienta visual y gráfica que nos permite crear una clase con sus respectivas propiedades, eventos y métodos.



2. Módulo de clases

   Es la forma de construir las clases, con sus respectivas propiedades, eventos y métodos utilizando el módulo de clase y escribiendo el código directamente.

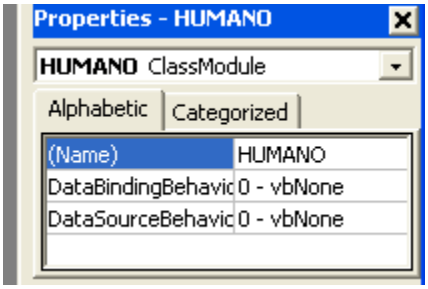
Para crear la clase es necesario agregar al proyecto en uso el módulo de clase; el cual se selecciona del menú Proyecto (**Class Module**).



Una vez activada la interfaz del módulo de clase, hay que escribir el código para crear la clase.

Creación de la clase

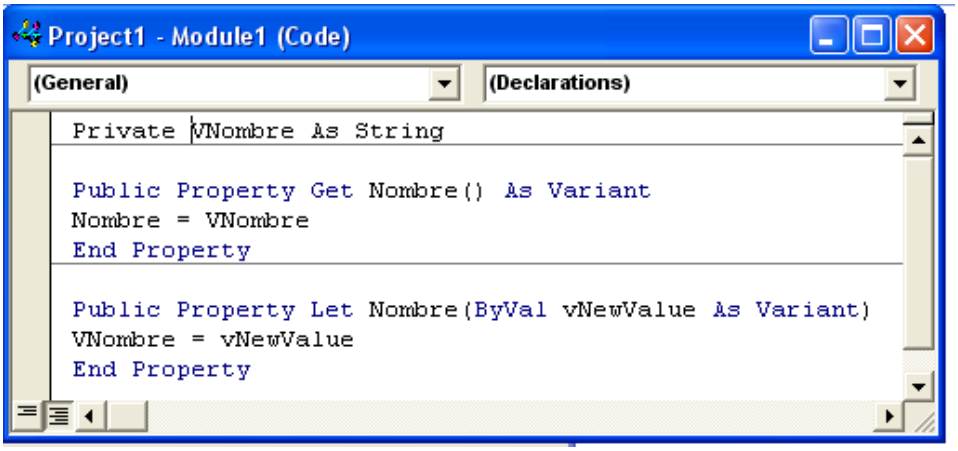
Una vez que se activa el módulo de clase, existe la clase, lo que compete ahora es nombrar a la clase. Para ello se debe manipular la propiedad **name** de la ventana de propiedades del módulo de clase.



Creación de propiedades

Para crear una propiedad se deben realizar los siguientes pasos:

1. Definir la propiedad declarando una variable privada del tipo de dato que la propiedad es capaz de manipular
2. Declarar una función de propiedad **Get**
3. Declarar un procedimiento de propiedad **Let**



Property Let es el procedimiento que permite asignar un valor a la propiedad y Property Get es el procedimiento que hace que la propiedad regrese el valor almacenado.

Creación de métodos

Los métodos representan la funcionalidad que tiene la clase, para crearlos se declaran procedimientos o funciones públicas en el módulo de clase.

La siguiente función es un ejemplo que crea un método que calcula un promedio:

Public Function Promedio(exam1 As Integer, exam2 As Integer) As Integer

Promedio = (exam1 + exam2) / 2

End Function

Creación de eventos

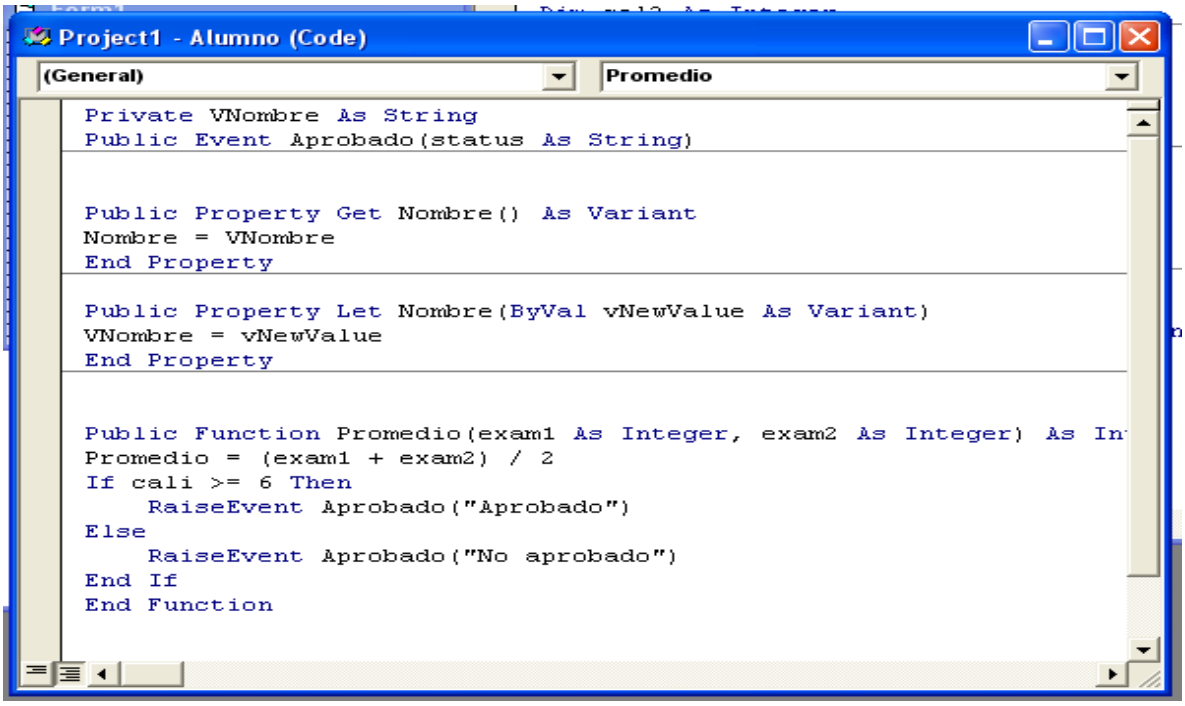
Con el uso de los eventos, la clase provee notificaciones de que una acción está ocurriendo. Para crear un evento se debe:

1. Declarar el evento:

Public Event *nombre del evento* (parámetros)

2. Disparar el evento:

RaiseEvent *nombre del evento.*



Una vez que se ha creado la clase, es posible utilizarla en una aplicación. La forma de utilizar la clase es creando la instancia de la clase (objetos) para tener acceso a las propiedades, eventos y métodos de la clase que se generó. La forma de realizar esta acción es la siguiente:

- Declaración de la variable de tipo clase:

  Dim *variable* As *Nombre de clase*

- Instanciación de la clase*:*

  Set *variable* = New *Nombre de clase*

Una vez creado el objeto, se procede a manipular sus elementos como propiedades, eventos y métodos.

4.2 Data Bound Class Module

Una clase es un objeto que encapsula datos y código, y las propiedades de una clase son los datos que describen un objeto.

Para trabajar con orígenes externos de datos, se debe hacer una clase de reconocimiento de datos. Las clases de reconocimiento de datos se pueden dividir en dos categorías: receptoras de datos y orígenes de datos. Los módulos de clase tienen dos propiedades en tiempo de diseño, *DataBindingBehavior* y *DataSourceBehavior,* que determinan cómo interactuará una clase con los datos externos. El objeto *BindingCollection* se usa para enlazar las clases de reconocimiento de datos con controles o entre sí.

Orígenes de datos

Un origen de datos es una clase que proporciona datos de una fuente externa para que los reciban otros objetos. Un *control Data* es, en realidad, una instancia de una clase que es un origen de datos. A diferencia del control Data, una clase de reconocimiento de datos no necesita tener una representación visual y no está limitado a una interfaz de datos en particular, como *Data Access Objects (DAO)* o *Remote Data Objects (RDO).* De hecho, una clase de reconocimiento de datos puede actuar como origen de datos para cualquier tipo de datos, incluyendo los orígenes ODBC tradicionales, ActiveX Data Objects (ADO) o cualquier proveedor de bases de datos (DB) OLE.

La propiedad *DataSourceBehavior* determina si una clase puede actuar como origen de datos. Al establecer *DataSourceBehavior* a 1 *(vbDataSource*), su clase puede actuar como origen de datos para otros objetos.

Receptores de datos

Un receptor de datos es una clase que se puede enlazar a un origen externo de datos, parecido a cómo un control *TextBox* se puede enlazar a un control Data. En versiones anteriores de Visual Basic, los controles eran los únicos objetos que se podían enlazar a un origen de datos. Las clases de reconocimiento de datos establecidas como receptores de objetos le permiten enlazar cualquier objeto a cualquier otro objeto creado desde una clase que se haya establecido como origen de datos.

La propiedad *DataBindingBehavior* permite a una clase enlazar a datos externos.

Las dos opciones de esta propiedad son los siguientes:

- 1 *(vbSimpleBound)*, un objeto creado desde su clase estará enlazado a un único campo de datos en un origen externo de datos.
- 2 (*vbComplexBound)*, su clase estará enlazada a una fila de datos en un origen externo de datos. Visto de este modo, si sus objetos fueran controles, un control TextBox estaría enlazado de forma sencilla, mientras que un control de cuadrícula lo estaría de forma compleja.

Objeto BindingCollection

Del mismo modo que enlazaría un control a una base de datos mediante un control Data, las clases de reconocimiento de datos necesitan un objeto central que las enlace. Ese objeto es *BindingCollection*, consiste en una colección de enlaces entre un origen de datos y uno o más receptores de datos.

Para usar el objeto *BindingCollection* se debe agregar primero una referencia a *Microsoft Data Binding Collection* seleccionándola en el cuadro de diálogo ‘Referencias’, disponible en el menú ‘Proyecto’. Como con cualquier otro objeto, se necesitará crear una instancia del objeto *BindingCollection* en tiempo de ejecución.

La propiedad *DataSource* del objeto *BindingCollection* se usa para especificar el objeto que proporcionará los datos. Este objeto debe ser una clase o *UserControl* con su propiedad *DataSourceBehavior* establecida a *vbDataSource*.

> Una vez que se ha instanciado *BindingCollection* y se ha establecido su DataSource, se puede usar el método Add para definir las relaciones del enlace. El método Add toma tres argumentos requeridos: el nombre del objeto receptor, la propiedad de ese objeto que se va a enlazar con el origen y el campo del origen que se enlazará a la propiedad.[6]

Bibliografía del tema 4

Budd, Timothy, (1994) *Introducción a la programación Orientada a Objetos*, México, Addison Wesley.

Ceballos, Francisco Javier, (2000), *Enciclopedia de Microsoft Visual Basic 6*, México, Alfaomega-Rama.

Eduardo Olaz, *Visual Basic. Manual del programador. Qué se puede hacer con Visual* Basic, material en línea, disponible en: http://www.olaz.net/descargas/manuales/programarconobjetos.pdf.

Vaughn, William R., (2000), *Visual Basic and SQL Server,* Microsoft Press, EE.UU.

[6] Eduardo Olaz, *Visual Basic. Manual del programador. Qué se puede hacer con Visual* Basic, material en línea, disponible en: http://www.olaz.net/descargas/manuales/programarconobjetos.pdf. pp. 67-68. Fecha de recuperación: 9 de diciembre 2008.

Actividades de aprendizaje

A.4.1 Elaborar (en una aplicación en Visual Basic) un programa que genere una clase con el nombre HUMANO que contenga las propiedades NOMBRE, EDAD; un método RESPIRAR y un evento CORRER. Ver anexo 4.1.

A.4.2 Generar una “ClaseOrigen” que permita en una aplicación con un formulario, acceder a los datos contenidos en una tabla. Ver anexo 4.2.

A.4.3 Elabora una clase Alumno con las siguientes características

Propiedades	Eventos	Métodos
Nombre	Acreditar	CalcularPromedio
Carrera	Estudiar	DespliegaDatos

Cuestionario de autoevaluación

1. ¿Cómo defines una clase?
2. ¿Qué es una instancia de la clase?
3. ¿Qué es el módulo de clases?
4. Enlista los pasos para crear una propiedad.
5. Al declarar propiedades, ¿cuál es el procedimiento de lectura y cuál es el de escritura?
6. ¿Cómo se crean los métodos de una clase?
7. ¿Qué representan los eventos?
8. ¿Qué es una clase origen?
9. ¿Qué propiedad se modifica para que una clase sea capaz de acceder a datos?
10. Indica qué propiedades tienen los módulos de clase que permiten determinar cómo interactúa una clase con datos externos.

Examen de autoevaluación

Elige la opción correcta

1. Definición formal de un tipo objeto

a) Clase

b) Agente

c) Variable

d) Tipo de dato

e) Atributos

2. Nombre de la acción que consiste en hacer una copia funcional de la clase

a) Objeto

b) Instanciación

c) Heredar

d) Creación

e) Copiar

3. Son un conjunto de datos y comportamientos

a) Algoritmo

b) Base de datos

c) Clase

d) Métodos

e) Eventos

4. Procedimiento que permite asignar un valor a una propiedad

a) Property Let

b) Property Pet

c) Property Get

d) Property Asignar

e) Property Propiedad

5. Procedimiento que hace que la propiedad regrese un valor almacenado

a) Property Let

b) Property Pet

c) Property Get

d) Property Asignar

e) Property Propiedad

6. Representan la funcionalidad que tiene una clase y se crean declarando procedimientos o funciones

a) Property Set

b) Procedimientos de propiedad

c) Funciones de propiedad

d) Métodos

e) Eventos

7. Herramienta utilizada para construir clases, con sus respectivas propiedades, eventos y métodos escribiendo el código directamente:

a) Módulo de acceso a clases

b) Class Module

c) Class Builder

d) Módulo de mantenimiento de clases

e) Object Browser

8. Sintaxis utilizada para instanciar un objeto

a) Dim objeto as clase

b) Set objeto as clase

c) Set clase = objeto

d) Set objeto as new clase

e) Set objeto = new clase

9. Objeto que se usa para enlazar las clases de reconocimiento de datos con otros controles o entre sí

a) Data Behaviour

b) DataSource Behaviour

c) BindingCollection

d) Dat SourceCollection

e) DataBindingCollection

10. Propiedad que permite a una clase enlazar a datos externos

a) Data Behaviour

b) DataSource Behaviour

c) BindingCollection

d) Dat SourceCollection

e) DataBindingBehavior

TEMA 5. ACTIVE-X Y CONTROL-Y COMPONENT OBJECT MODEL

Objetivo particular

El alumno reconocerá las ventajas de utilizar ActiveX Control y la tecnología COM, e identificará la forma de utilizar Microsoft Internet Explorer aplicando la tecnología COM en la creación de un proyecto.

Temario detallado

5.1. Introducción a Active-X Control
5.2. Construcción de aplicaciones con Active-X Control
5.3. Introducción a Component Object Model
5.4. Construcción de aplicaciones con COM
5.5. Uso de Microsoft Internet Explorer

Introducción

OLE es un término familiar cuando se programa en Windows, significa vinculación e incrustación de objetos. Esta tecnología se ha convertido en una arquitectura que define otras muchas tecnologías y diseños de programación. Un ejemplo de estas tecnologías son COM (Component Object Model) y ActiveX.

5.1 Introducción a Active-X Control

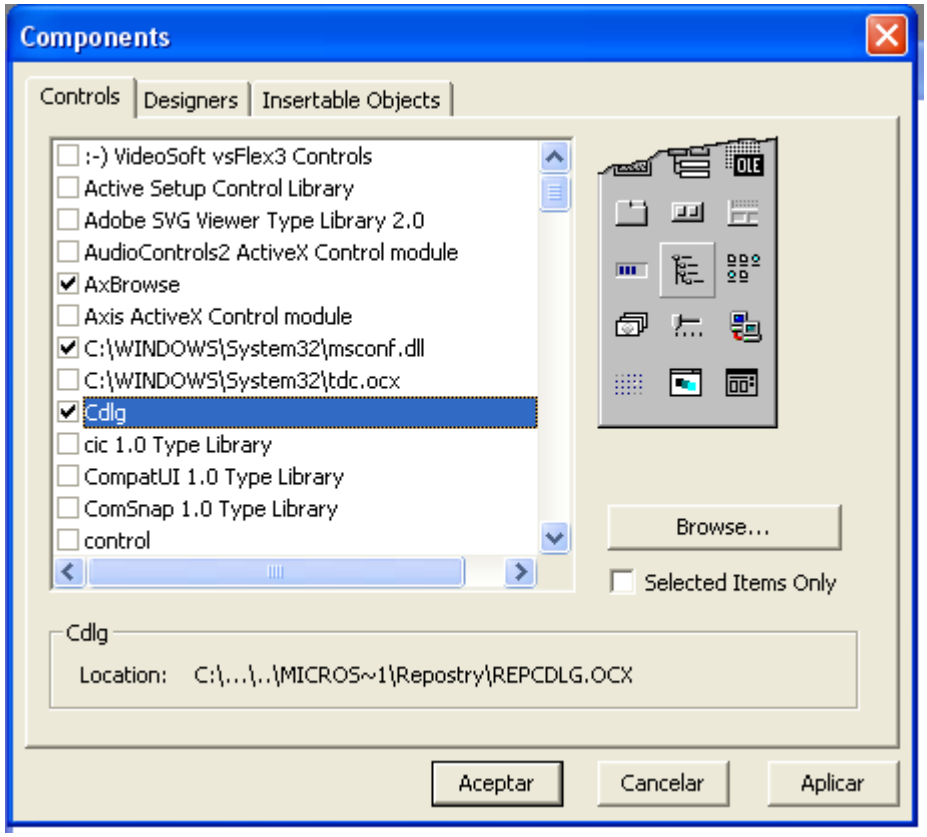
La tecnología COM y Active-X son la versión actual del conjunto de controles que comenzó hace algunos años con los controles VBX. Los controles Active-X comenzaron como controles que extendían a Visual Basic, que se podían añadir para aumentar la cantidad de herramientas del cuadro de herramientas.

En ese entonces los controles no eran compatibles con exploradores web y otras herramientas de programación para Windows como el Visual C++. Debido a la popularidad de los controles VBX, Microsoft se vio forzado a rediseñarlos para que fuesen aprovechados por cualquier herramienta.

Microsoft presentó los controles OCX como una evolución a la tecnología VBX. Estos controles extendieron VB a 32 bits así como a otros lenguajes como Visual C++. Los controles OCX no pueden transmitirse con facilidad a través de Internet. Por eso Microsoft mejoró para que fueran controles ActiveX y facilitasen su transmisión e integración con los exploradores web que soportan esta tecnología, así como con varias aplicaciones y lenguajes de programación.

Un control ActiveX es un objeto reusable que incluye elementos visuales y código que no forman parte del conjunto de controles estándar de Visual Basic, que se pueden añadir para aumentar la cantidad de herramientas del cuadro de herramientas y usar para desarrollar programas. Estos controles han sido desarrollados generalmente por terceras personas. Existen como archivos independientes con extensión OCX y deben ser cargados antes de utilizarse.

La forma de añadir controles ActiveX al cuadro de herramientas es seleccionando la opción de componentes del menú Proyecto, esta acción abrirá una ventana con la lista de todos los controles disponibles para su selección.



Cuando se añade un control ActiveX al cuadro de herramientas para utilizarlo, se manipulan los siguientes elementos:

- Propiedades
- Eventos
- Métodos

Es decir, un control ActiveX se usa como cualquier otro control.

5.2. Construcción de aplicaciones con Active-X Control

Cuando se crea un control Active-X, se conoce como un control de clases, las cuales actúan como una plantilla para diseñar el control. El código y valores de propiedades se guardan en un archivo con extensión .ctl. Un Proyecto Active-X

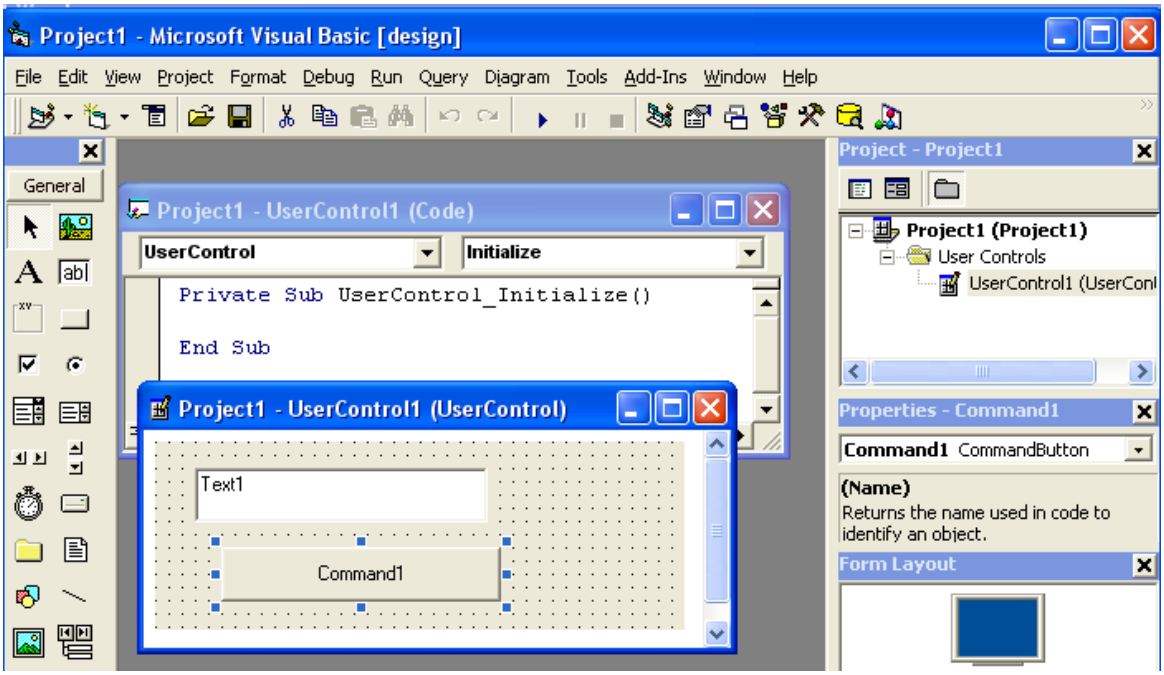
Control puede contener uno o más archivos .ctl y cuando se compila, genera un archivo .ocx.

Para construir un Active-X Control, se siguen los siguientes pasos:

1. Crear un Active-X Control Project
2. Crear la interface de usuario para el control
3. Crear propiedades, eventos y métodos
4. Crear procedimientos de eventos para el control
5. Exponer eventos para el control
6. Crear páginas de propiedad para el control
7. Verificar y probar el control.[7]

La interfaz que se utiliza para diseñar el Active-X se llama UserControl, cada objeto que se incrusta para la creación del control se conoce como control constituyente, y una vez que el diseño del control se coloca en un formulario, se crea una instancia del control.

[7] Cf. el Blog de Compupráctico, “Controles Active-X”, 01/10/08, disponible en: http://computopractico.blogspot.com/2008/09/controles-activex.html. Fecha de recuperación: 09 de diciembre de 2008.



En VB, los controles son objetos reusables que incluyen elementos visuales y código. Se conocen como un Control Class, el cual actúa como plantilla para un control. Cuando se compila se genera un archivo .ocx

5.3 Introducción a Component Object Model

COM es un resultado de la tecnología OO. COM (Component Object Model), es una especificación que se basa en un estándar binario para generar módulos de código pre-compilados (conocidos como componentes) reusables, de fácil mantenimiento y modificación, COM determina la manera como un programa accede a los componentes y los utiliza. El lenguaje de programación utilizado puede ser Visual Basic, C ++, etc.

> COM utiliza código a nivel de sistema que se ejecuta en forma de librerías de enlace dinámico, que son llamados colectivamente COM biblioteca. La biblioteca consta de varias interfaces de programación de aplicaciones (API) de las funciones que son necesarias para componentes COM que hacen diversas tareas. COM biblioteca se encarga de localizar y activar

aplicaciones de servidor. Con las aplicaciones cliente COM API contiene las funciones necesarias para instanciar objetos que el cliente desee utilizar.[8]

El uso de la tecnología COM permite que un objeto se utilice por su funcionalidad, independientemente del lenguaje o herramienta empleado para crearlo. Con COM existe una forma estándar de comunicarse con los objetos sin importar si se usará en Visual Basic, Visual C++, Visual Java++, entre otros.

Características de COM

- Encapsulamiento
- Descubrimiento dinámico, enlace tardío
- Procesamiento de transacciones
- Cooperación distribuida
- Seguridad integrada
- Sólo pensado para Windows, en otras plataformas a través de productos de terceros.

Clasificación de COM:

- COM95: propuesta original
- DCOM (COM97): en redes
- COM+DCOM= de "bajo nivel"
- OLE (1995), ActiveX (1997), MTS (1999) son "de nivel de aplicación", "alto nivel", construidas sobre COM & DCOM

[8] *COM uses system-level code that is implemented in the form of dynamic link libraries, which are collectively called COM library. The COM library consists of several application programming interface (API) functions that are necessary for COM components to do various tasks COM library is responsible for locating and activating server applications. With client applications, COM APIs contains the necessary functions for instantiating objects that the client wants to use*. Véase "Component Object Model", material electrónico en línea, disponible en: http://www.peterindia.net/COMOverview.html, la traducción es mía. Fecha de recuperación 09/12/08.

COM - OLE

- Provee servicios de enlace e incrustación de objetos
- Creación de documentos compuestos

COM - ActiveX

- Extiende capacidad básica de COM
- Permite incrustar componentes en sitios web
- Permite incrustar componentes en otros objetos

COM - MTS

- Microsoft Transaction Server
- Son servicios basados en COM para soluciones empresariales
- Coordinación de transacciones
- Seguridad
- Permite construir sistemas de información empresariales sobre COM

COM+

- Integra servicios de MTS con cola de mensajes en COM
- Lenguajes soportados
  - Visual Basic
  - Visual C++
  - Java++
- Encapsula cierta complejidad, facilita el uso de COM

Ventajas de COM

1. COM promueve componentes basados en el desarrollo de software y viene con una serie de ventajas, como la capacidad de utilizar pre-empaquetados a componentes y herramientas de proveedores terceros en una aplicación y el apoyo a la reutilización de código en otras partes de la misma aplicación.

2. COM promueve la reutilización de código. Los estándar de las clases normalmente son reutilizados en la misma solicitud, pero no al ser usada fácilmente en otras aplicaciones.
3. COM promueve la programación orientada a objetos (POO). Las principales características de la POO son encapsulado, que permite a los detalles de la implementación de un objeto ocultarse, polimorfismo, que es la capacidad de exposición de múltiples comportamientos, y la herencia, que permite la reutilización de las clases existentes, a fin de diseñar nuevas y más especializadas clases.
4. COM comprende los mecanismos necesarios para comunicarse entre sí. En el caso normal, dos componentes codificados usando dos diferentes lenguajes de programación no pueden comunicarse entre sí. Pero COM puede hacer posible que los distintos componentes de lenguajes, que se adhieren a la especificación COM, puedan interactuar unos con otros y, por lo tanto COM es independiente del lenguaje.
5. COM ayuda para acceder a los componentes cargados en diferentes máquinas en la red. Aplicaciones que utilizan COM consiguen acceder y compartir componentes COM, independientemente de su ubicación. Así COM proporciona la transparencia y la ubicación de los componentes COM que son independientes de la ubicación.[9]

5.4. Construcción de aplicaciones con COM

---

[9] *1. COM promotes component-based software development - before component-based development came, software programs have been coded using procedural programming paradigm, which supports linear form of program execution.*
2. COM promotes code reusability - standard classes are normally reused in the same application but not to be used easily in other applications.
3. COM promotes Object-oriented programming (OOP) - The primary characteristics of OOP are encapsulation, which allows the implementation details of an object to be hidden, polymorphism, which is the ability to exhibit multiple behaviors, and inheritance, which allows for the reuse of existing classes in order to design new and more specialized classes.
4. COM comprises the necessary mechanisms for COM components to communicate with each other - in the normal case, two components coded using two different programming languages cannot communicate with each other. But COM can make it possible for different language components, which adhere to the COM specification, to interact with each other and hence COM is language-independ.
5. COM helps to access components loaded in different machines on the network - COM component can reside anywhere on our computer or even on another computer connected to a network. That is, applications using COM can access and share COM components regardless of their location. Thus COM provides location transparency and COM components are location independen. "Component Object Model: Advantages", material electrónico disponible en: http://www.peterindia.net/COMOverview.html, la traducción es mía. Fecha de recuperación 09/12/08.

Un componente COM es una unidad de código que provee cierta funcionalidad. Generalmente son objetos ejecutables, como un archivo .exe, .dll, o .ocx que sigue las especificaciones para proveer objetos. Un componente para el desarrollo de software se basa en construcciones fundamentales de los paradigmas orientados a objetos. “Aunque los términos "componente" y "objeto" suelen utilizarse indistintamente, un componente no es un objeto. Un objeto es una instancia de una clase que se crea en tiempo de ejecución. Un componente puede ser una clase, pero por lo general es una colección de clases e interfaces. Un componente COM expone objetos que pueden ser usados por otras aplicaciones.”

Se pueden crear tres tipos de componente con VB:

- ActiveX Controls. Elementos de interfaces de usuario estándar que permiten reunir formas reusables y cajas de diálogos.
- ActiveX Documents. Son aplicaciones que pueden ser hospedadas en un explorador para Internet.
- Programas COM ejecutables y DLLs. Programas COM ejecutables y DLLs son librerías de clases. Las aplicaciones usan objetos COM al hacer las instancias de clases provistas por COM archivos .exe o .dll. La aplicación llama a las propiedades, eventos y métodos proporcionados por cada objeto COM.

5.5 Uso de Microsoft Internet Explorer

Usando Visual Basic se pueden crear aplicaciones que desplieguen páginas HTML; con el uso de Controles ActiveX se pueden crear documentos que pueden hospedar a Internet Explorer o cualquier otro documento componente y también, se pueden crear aplicaciones web que combinan la seguridad y conveniencia de componentes COM con la flexibilidad del diseño de páginas, es decir, se puede escoger cualquiera de estos procesos para que operen en un cliente web de un servidor Internet. A continuación se analiza un poco más cada opción.

Controles ActiveX

Se puede tener acceso a Internet desde las aplicaciones, VB es una herramienta que se adapta muy bien para su uso en Internet. Para tener acceso se pueden utilizar varios controles ActiveX basados en Internet que encapsulan o empaquetan la aplicación existente y la codifican en aplicaciones orientadas a Internet.

Los controles que encapsulan las aplicaciones para que puedan funcionar con la tecnología Internet son:

- Control de transferencia Internet

  Encapsula los tres protocolos de Internet más populares: HTTP, FTP (Protocolo de transferencia de archivos) y Gopher (un protocolo de búsqueda para ayudar a buscar información en Internet).[10]

- Control WebBrowser

Agrega capacidades de búsqueda, vista de documentos y descargas de datos a las aplicaciones Visual Basic. Da a los usuarios la habilidad de consultar sitios web o carpetas en un sistema local de archivos o de red.

- Control WinSock

Da una conexión común de Windows y un control de intercambio de datos que proporciona dos protocolos: UPD (Protocolo de Datagrama del Usuario) y TCP (Protocolo de Control de Transferencia).

---

[10] Greg M. Perry, *Aprendiendo Visual Basic 6 en 21 días*, México, Pearson Education, 2000, p. 617, disponible también en formato digital en: http://books.google.com.mx/books?id=QMmQTUT0XMcC&pg=RA3-PA612&lpg=RA3-PA612&dq=Encapsula+los+tres+protocolos+de+Internet+m%C3%A1s+populares:&source=bl&ots=kiZM9Fbx4I&sig=dDbj_IXfmX7QD5sABiX3BoGzws8&hl=es&sa=X&oi=book_result&resnum=10&ct=result#PPR3,M1. Fecha de recuperación: 09 de diciembre de 2009.

ActiveX Document

Si se desea desarrollar una aplicación VB basada únicamente para Internet, se puede hacer utilizando documentos ActiveX. Un documento ActiveX actúa y es parecido a cualquier aplicación normal en un formulario, con excepción de que este Documento ActiveX envía controles ActiveX a la computadora del usuario final si es que esa computadora no tiene los controles ActiveX que usa el documento. Este documento llega al usuario y luce como una página web basada en HTML normal.

La razón más importante para crear un Documento ActiveX es que Internet Explorer puede ejecutarlo como si fuera un programa de control o para ejecutar programas del sistema operativo. Los menús del documento ActiveX se intercalan con los del Internet Explorer.[11]

Aplicaciones Internet de VB

VB puede crear dos tipos de aplicaciones Internet:

- Aplicaciones IIS

Es la forma más sencilla para incorporar Internet en las aplicaciones es hacerlo como una aplicación de Internet Information Server (IIS).[12] Es crear aplicaciones que se procesan en un servidor, y es accesible desde un web browser o cualquier plataforma. Consiste en un componente COM y un simple Active Server Page (archivo .asp). ASP Páginas Active Server (ASP, Active Server Pages) es un entorno para crear y ejecutar aplicaciones dinámicas e interactivas en la Web.

Se puede combinar páginas HTML, secuencias de comandos y componentes ActiveX para crear páginas y aplicaciones web interactivas. El componente COM

[11] Véase, Greg M. Perry, op. cit., p. 615.
[12] ibid, p. 624.

contiene la funcionalidad de la aplicación, y el archivo .asp existe al instanciar el componente COM en el servidor web.

- Aplicaciones DHTML

Consisten en una o más páginas HTML y un componente COM que interactúa con la página. La página HTML actúa como la interface de usuario para la aplicación y el componente COM contiene la funcionalidad para la aplicación. Permiten escribir código que responde a eventos en una página web HTML. El explorador web del usuario final interpreta y realiza estos comandos para que el servidor remoto tenga poco qué hacer, a excepción de responder a peticiones especiales cuando se presentan, como la recuperación de páginas web adicionales.[13]

[13] loc. cit.

Bibliografía del tema 5

Ceballos, Francisco Javier, (2004), *Curso de programación de Visual Basic 6,* 2ªed., México, Alfaomega-Rama.

_____, (2000), *Enciclopedia de Microsoft Visual Basic 6*, Alfaomega-Rama, México.

Iqbal, Kamran y Tony Jamieson, (1999), *Microsoft Visual Basic 6.0 Fundamentals*, USA, Microsoft Press.

Greg M. Perry, (2000), *Aprendiendo Visual Basic 6 en 21 días*, México, Pearson Education.

Actividades de aprendizaje

A.5.1 Describe las ventajas del uso de componentes COM y controles ActiveX.

A.5.2 Genera un control ActiveX y pruébalo en un formulario. Ver anexo 5.1.

A.5.3 Indica las características de COM y su relación con los 4GL.

A.5.4 Elabora un cuadro comparativo entre los tipos de aplicaciones Internet.

Cuestionario de autoevaluación

1. ¿Qué es un control ActiveX?
2. ¿Cuáles son los pasos a seguir en la construcción de controles ActiveX?
3. ¿Qué es el Component Object Model (COM)?
4. ¿Cómo se clasifican los COM?
5. ¿Qué es un componente?
6. ¿Cuál es la diferencia entre un componente y un objeto?
7. ¿Cuáles son los tres tipos de componentes que se pueden crear en Visual Basic?
8. ¿Qué controles encapsulan aplicaciones para que puedan funcionar con Internet?

9. ¿Qué razón justifica la creación de ActiveX Document?

10. ¿Cuáles son los tipos de aplicaciones internet que se pueden generar en Visual Basic?

Examen de autoevaluación

Elige la respuesta correcta

1. Término familiar cuando se programa en Windows, significa vinculación e incrustación de objetos

a) Controles ActiveX

b) Controles OCX

c) *OLE*

d) *ASP*

e) *Controles EXE*

2. Controles que facilitan la transmisión e integración con los exploradores web:

a) Controles ActiveX

b) Controles OCX

c) Controles básicos

d) Controles VBX

e) Controles EXE

3. Archivo que se genera cuando se compila un control ActiveX:

a) .exe

b) .com

c) .ctl

d) .ocx

e) .dll

4. Especificación que se basa en un estándar binario para generar módulos de código pre-compilados (conocidos como componentes) reusables:

a) Objeto

b) Clase

c) COM

d) API

e) Control Class

5. Extensión del archivo que guarda el código y valores de propiedades de un Control ActiveX

a) .exe

b) .com

c) .ocx

d) .ctl

e) .dll

6. Nombre de la interfaz que se utiliza para crear un ActiveX

a) UserActiveX

b) UserControl

c) UserCom

d) UserDll

e) UserActive

7. Son características de COM

a) Encapsulamiento y Cooperación distribuida

b) Propiedades y Eventos

c) Encapsulamiento y Desarrollo

d) Encapsulamiento y Propiedades

e) Eventos, Propiedades y encapsulamiento

8. Tipo de componente COM que consiste en aplicaciones que pueden ser hospedadas en un explorador para Internet

a) ActiveX Control

b) Com dll

c) ActiveX Internet

d) Com Documents

e) ActiveX Documents

9. Es crear aplicaciones que se procesan en un servidor, y es accesible desde un web browser o cualquier plataforma

a) Aplicación WEB

b) Aplicación DHTML

c) Aplicación ISS

d) Aplicación IIS

e) Aplicación DTML

10. Tipo de aplicación donde una página HTML actúa como la interface de usuario y el componente COM contiene la funcionalidad

a) Aplicación WEB

b) Aplicación DHTML

c) Aplicación ISS

d) Aplicación IIS

e) Aplicación DTML

11. Entorno para crear y ejecutar aplicaciones dinámicas e interactivas en la Web

a) ASP

b) COM

c) ActiveX

d) HTML

e) DHTML

TEMA 6. OPTIMIZACIÓN Y DISTRIBUCIÓN DE APLICACIONES

Objetivo particular

El alumno describirá la variedad de técnicas que optimizarán el rendimiento de una aplicación y su distribución con la construcción de un programa setup.

Temario detallado

6.1 Optimizando una aplicación

6.2 Distribución de una aplicación

6.3 Distribución de Active-X Control

Introducción

Antes de distribuir una aplicación a los usuarios es conveniente mejorar la funcionalidad, rapidez, rendimiento, incluir documentación, seguridad de los recursos involucrados, etc., y para distribuir esta aplicación, no es solamente distribuir el programa .exe, sino también todos los archivos que sirven de soporte para la aplicación, como archivos .dll o controles.

6.1 Optimizando una aplicación

Cuando se optimiza una aplicación se busca que sea tan efectiva como sea posible. Se busca reducir el tiempo de ejecución, velocidad y cantidad de memoria, afinar o realzar el desempeño y espacio en memoria.

A continuación se describen algunas técnicas que ayudan a optimizar una aplicación.

Aumentar velocidad

Se puede aumentar la velocidad de las formas siguientes:

- Cargando formularios previamente.
- Almacenando gráficos como mapa de bits.
- Utilizando rutinas de biblioteca de vínculos dinámicos.
- Utilizando enteros, variables largas en vez de único, doble o variables de moneda.

Aumentar recursos disponibles

Se puede aumentar recursos disponibles de las formas siguientes:

- Creando controles simulados utilizando un objeto gráfico.
- Dibujando imágenes gráficas durante tiempo de ejecución.
- Utilizando el control *Image* en lugar de cuadros de imagen.

Aumentar RAM disponible

Se puede aumentar RAM disponible de las formas siguientes:

- Utilizando variables de entero siempre que sea posible.
- Creando matrices dinámicas para liberar matrices cuando no son necesarias.
- Colocando o descargando controles y formularios cuando no es necesario.
- Utilizando variables locales.

Aumentar espacio en disco

Se puede aumentar espacio en disco de las formas siguientes:

- Generando controles en tiempo de carga.
- Minimizando el tamaño de encabezado.
- Eliminando funciones y subrutinas innecesarias.
- Eliminando objetos no utilizados y métodos asociados.

6.2 Distribución de una aplicación

Cuando se distribuye una aplicación se debe proporcionar al usuario con un programa instalador que desarrolle las siguientes actividades:

- Copiar los archivos necesarios que el usuario va a requerir en su computadora.
- Colocar los archivos en el directorio apropiado.
- Registrar archivos.
- Crear un elemento o grupo en el menú Inicio.
- Crear un icono para el escritorio del usuario.

Todas estas actividades se pueden realizar con un asistente de empaquetado y distribución (Package and Deployment Wizard) que incluye las siguientes tareas:

- Compila la aplicación y comprime los archivos.
- Crea un programa de instalación que se utiliza para instalar la aplicación.
- Determina la mejor organización para los discos flexibles de instalación, crea numerosos discos de instalación y divide y organiza los archivos grandes en varios discos flexibles.
- Copia la aplicación compilada a un disco duro para que se pueda instalar la aplicación a través de una red o un quemador de CD-ROM.
- Prepara la aplicación para su distribución a través de Internet.
- El asistente de empaquetado genera una lista de los archivos que se necesitan para la instalación. Una aplicación requiere archivos DLL y OCX, los cuales deben residir en el área de instalación destino con el programa compilado y el programa de instalación.

6.3 Distribución de ActiveX Control

La forma de distribuir un control depende de cómo se usará el control. Un ActiveX Control puede usarse en diferentes tipos de aplicaciones, como Microsoft Office o

aplicaciones en Visual Basic, o como parte de un sitio web. Se puede distribuir el control de dos formas:

1. Como un componente compilado

Para distribuir un control compilado (archivo .ocx) con una aplicación, se debe crear el Control ActiveX como clase pública. De esta forma los usuarios del control pueden incluir el control compilado en el programa SetUp de su aplicación.

Las ventajas de distribuir un control compilado son:

- Se guarda la implementación del control confidencialmente.
- Los usuarios del control no pueden cambiar la implementación del control.
- Se pueden realizar pequeños arreglos o destacadas actualizaciones al control y cambiar el reciente control compilado a todos los usuarios del control.

2. Como código fuente formando parte de una aplicación.

- Se puede incluir cualquier archivo .ctl en cualquier Proyecto de Visual Basic. Cuando la aplicación se compila, el código fuente del control se compila como parte del archivo ejecutable de la aplicación.

Las ventajas de distribuir un control como código fuente son:

- No existe un archivo .ocx para distribuir.
- La depuración es más fácil porque sólo se necesita depurar partes específicas de la aplicación.

Las desventajas de distribuir controles como código fuente son:

- Pequeños arreglos en el código del control requiere que se compile toda la aplicación.

- Múltiples aplicaciones requieren espacio en disco adicional porque cada aplicación incluye todo el código fuente del control.
- Cada vez que se utilice el código fuente del control, será una oportunidad para arreglarlo o actualizar el código. Esto puede complicar el guardar pistas o versiones de cuál control fue usado en cuál versión o en cuál aplicación.

Bibliografía del tema 6

Ceballos, Francisco Javier, (2004), *Curso de programación de Visual Basic 6,* 2ª Edición, México, Alfaomega-Rama.

_____, (2000), *Enciclopedia de Microsoft Visual Basic 6*, Alfaomega-Rama, México.

Actividades de aprendizaje

A.6.1 Utilizando las técnicas de optimización, mejore la aplicación realizada en el anexo 4.2 para aumentar memoria RAM y velocidad.

A.6.2 Utilizando el asistente de empaquetado y distribución (Package and Deployment Wizard), genera un programa instalador (set up) para la práctica 4.1 localizada en el Anexo y otro programa instalador (set up) para la práctica 5.1 del anexo que permitan la instalación de dichas prácticas en cualquier equipo de cómputo.

Cuestionario de autoevaluación

1. ¿Qué aspectos se buscan cuando se optimiza una aplicación?
2. Cuando se optimiza una aplicación, ¿qué técnicas son de ayuda?

3. ¿Cómo se puede aumentar la velocidad de una aplicación?
4. ¿Qué tipos de datos es recomendable utilizar para aumentar la memoria RAM?
5. ¿Qué actividades debe desarrollar un programa de instalación?
6. ¿Qué tareas puede realizar un asistente de empaquetado?
7. ¿Cuál es la diferencia entre empaquetar una aplicación y un control ActiveX?
8. ¿Cuándo se distribuye un control ActiveX como un componente compilado?
9. ¿Cuándo se distribuye un control ActiveX como código fuente formando parte de una aplicación?
10. ¿Qué ventajas tiene el distribuir un control ActiveX como código fuente?

Examen de autoevaluación

Elige la opción correcta

1. Acciones como cargar formularios previamente, almacenar gráficos como mapa de bits y colocar en un módulo independiente, rutinas de depuración, son técnicas utilizadas para aumentar:

a) Memoria RAM
b) Memoria Caché
c) Procesador
d) Velocidad
e) Espacio en disco

2. Generar controles en tiempo de carga, minimizar tamaño de encabezado y eliminar funciones y subrutinas innecesarias, son actividades que permiten

a) Aumentar memoria RAM
b) Disminuir memoria Caché
c) Aumentar procesador
d) Disminuir velocidad
e) Aumentar espacio en disco

3. Dibujar imágenes gráficas durante tiempo de ejecución, es una técnica de depuración. ¿Cuál es su objetivo?

a) Aumentar memoria RAM

b) Disminuir memoria Caché

c) Aumentar recursos

d) Disminuir velocidad

e) Aumentar espacio en disco

4. Copiar los archivos necesarios que el usuario va a requerir en su computadora y colocarlos en el directorio apropiado, son actividades que debe de realizar un programa. ¿Cuál es ese programa?

a) Programa de depuración

b) Programa ejecutable

c) Programa inicio

d) Programa de instalación

e) Programa fuente

5. Nombre de la herramienta que permite crear un programa de instalación de una aplicación

a) Programa de depuración

b) Asistente de empaquetado y distribución

c) Asistente de instalación

d) Programa de instalación

e) Programa de distribución

6. Ejemplos de aplicaciones donde se puede usar un control ActiveX

a) Aplicaciones Microsoft Office, Visual Basic o en un sitio web

b) Aplicaciones de empaquetado y distribución

c) Aplicaciones de asistente de instalación en Web

d) Aplicaciones de instalación de Microsoft Office

e) Aplicaciones de distribución de Visual Basic o en Web

7. Tipo de distribución de un control ActiveX que permite incluir el control en el programa SetUp de una aplicación

a) Como un control ActiveX compilado

b) Como un control ActiveX de empaquetado y distribución

c) Como un control ActiveX de instalación

d) Como un control ActiveX ejecutable

e) Como un control ActiveX editado

8. Guarda la implementación del control confidencialmente para que los usuarios del control no puedan cambiar la implementación del control, es una ventaja de distribuir un control bajo qué tipo de distribución

a) Como un control ActiveX ejecutable

b) Como un control ActiveX de empaquetado y distribución

c) Como un control ActiveX de instalación

d) Como un control ActiveX compilado

e) Como un control ActiveX editado

9. Forma de distribución de un control donde al compilar una aplicación, se compila el control como parte del archivo ejecutable de la aplicación

a) Como un control ejecutable

b) Como un control empaquetado y distribución

c) Como un código de instalación

d) Como un código objeto

e) Como un código fuente

10. Tipo de distribución de un control ActiveX que no es conveniente utilizar por presentar desventajas cuando se requiere compilar toda la aplicación al modificar el código del control ActiveX?

a) Como un control ejecutable

b) Como un control empaquetado y distribución

c) Como un código fuente

d) Como un código objeto

e) Como un código de instalación

Bibliografía básica

Ceballos Francisco Javier, (2004), *Curso de programación de Visual Basic 6,* 2ª Edición, México, Alfaomega-Rama.

_____, (2000), *Enciclopedia de Microsoft Visual Basic 6*, México, Alfaomega-Rama.

Franco Luis., Tutorial de ADO-1 Material en línea, disponible en: http://www.uyssoft.com pp. 1 Fecha de recuperación: 9 de diciembre 2008.

García de Jalón Javier, (1999), *Aprenda Visual Basic 6.0 como si estuviera en primero*, San Sebastián.

Iqbal, Kamran y Tony Jamieson. (1999), *Microsoft Visual Basic 6.0 Fundamentals*, EE.UU., Microsoft Press.

Perry Greg y Hettihewa Sanjaya, (2004), *Aprendiendo Visual Basic 6 en 24 horas*, México, Prentice Hall.

Révész György, E. (1991), *Introduction to formal languages,* Dover Publications.

Rob, Peter, *Sistemas de Bases de Datos*, México, Thomson, 2003.

Tucker, Allen, y Robert Noonan, (2003), *Lenguajes de Programación principios y paradigmas*, McGraw Hill.

Vaughn, William R., (2000), *Visual Basic and SQL Server,* EE.UU., Microsoft Press.

RESPUESTAS A LOS EXÁMENES DE AUTOEVALUACIÓN

	Tema 1	Tema 2	Tema 3	Tema 4	Tema 5	Tema 6
1	a	a	a	a	c	a
2	c	b	e	b	a	e
3	a	d	a	c	d	c
4	d	b	c	a	c	d
5	c	e	d	c	d	b
6	a	b	a	d	b	a
7	e	b	c	b	a	a
8	e	a	a	e	d	d
9	c	b	a	c	b	e
10	a	b	b	e	a	c