ESCUELA SUPERIOR POLITECNICA DEL LITORAL

**DEPARTAMENTO DE**
**INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

# " Análisis & Diseño ORIENTADO A OBJETOS

# de una Aplicación de GESTION COMERCIAL usando

# Arquitectura CLIENTE / SERVIDOR "

**TOPICO DE GRADUACION**
**Previa a la obtención del Título de:**
INGENIERO EN COMPUTACION

Presentado por:

LUIS A. JURADO F.

JOFFRE L. LEÓN V.

GUIDO M. MANTILLA D.

GUAYAQUIL - ECUADOR
1996

## *AGRADECIMIENTO*

Al Ing. Carlos Valero D., quien se mostró como un expositor claro, innato por los conocimientos que compartió con nosotros y nuestros demás compañeros en el Tópico de Graduación, así como por su dirección acertada en la ejecución de los Proyectos.
Esperamos explotar los conceptos que con altruismo impartió en nosotros y nos permitiremos tomar su ejemplo ó el que exige la vida misma, hacer de ésta un curso constante e intensivo de aprendizaje ...

Lo consideramos una persona valiosa en la enseñanza de la Computación a alto nivel y esperamos que lo siga dictando su cátedra por mucho más tiempo con el carisma que lo caracteriza ...

**INTRODUCCION** ...

El nacimiento de nuevos y poderosos procesadores para desktop ha sido de ayuda para el desarrollo evolutivo de Aplicaciones Distribuidas basadas en LAN.
Las aplicaciones que hoy pueden desarrollarse en estas plataformas sólo quedan limitadas a la imaginación de los diseñadores en vista de herramientas de Programación Visual que están en boga y van de la mano al Hardware disponible.
Cliente / Servidor:
" Un concepto estratégico global ",
" El presente & futuro de la Computación de negocios ",
" Arquitectura inteligente de alto nivel ",
... ó como Ud. lo encuentre en el encabezado de la revista de su preferencia, va a ser un tema que es y será titular en muchos ejemplares más.
Por ello nuestro estudio e investigación aplicando esta metodología en el desarrollo de la automatización de Gestiones Comerciales básicas que si bien es cierto están implementadas en miles de lenguajes, bajo diez mil plataformas, aún no es tan común escuchar que está diseñada con la filosofía que esta Tecnología implica.

DISEÑAR bajo Cliente / Servidor es un desafío estimulante, así como un paso más en el mundo de la Computación. Pero, eh ..! Cliente / Servidor va siempre mucho más allá de lo que de esquemas centralizados recuerde. Si Ud. desarrolla algo usando esta norma, acabado su trabajo cruze de brazos para pensar más, porque hay mucho más por crear y enlazar bajo este ambiente.

En Cliente / Servidor hay tareas para todos los gustos pues es un gran porcentaje de una técnica de programación, como también menor en una forma de implementación de Redes. Por qué tarea, se identifica más Usted ..?

Por cierto, estamos seguros que también ha escuchado sobre Polimorfismo, Herencia, Encapsulamiento & Information hiding, Clases, ... y los beneficios de estos conceptos. De seguro está pensando, otro paradigma en el mundo de la Computación ?. Pues, está en lo cierto y debido a sus beneficios (reusabilidad, fácil mantenimiento, productividad, ... ) también lo abordamos.

En definitiva, en este Análisis & Diseño estamos usando 2 metodologías en la Informática actual. Durante la elaboración de este trabajo pretendimos (dicho en forma burda) matar 2 pilluelos con una pedrada. Creemos haberlo conseguido, pero no los matamos porque vaya pilluelos !

# INDICE

CONTENIDO Pág.

## I. ANALISIS y DISEÑO DE UN SISTEMA DE GESTION COMERCIAL


1. OBJETIVOS DEL PROYECTO 1

2. DEFINICION DEL PROBLEMA 2

3. ANALISIS ESTRUCTURADO DE PROCESOS
3.1 Diagrama de Contexto de usuario combinado 3
3.2 Diagrama de Flujo de Datos de Nivel 1 4
3.3 Diagrama de Flujo de Datos de Nivel 2 5
3.3.1 Diagrama de Flujo de Datos de Nivel 2.2 6
3.4 Diagrama de Flujo de Datos de Nivel 3 7

4. ANALISIS ESTRUCTURADO DE DATOS:
Modelo Conceptual de Datos 8

5. DISEñO ESTRUCTURADO DE DATOS:
Modelo Físico de Datos

Tablas de la Base de Datos
Tabla: ARTÍCULOS 9
Tabla: ART_PROV 9
Tabla: BANCOS 9
Tabla: FAMILIA 10
Tabla: CLIENTES 10
Tabla: CTA_CTE 10
Tabla: CUENTAS 11
Tabla: DETALLE_ORDEN 11
Tabla: DOCUMENTOS 11
Tabla: CLIENTES 12
Tabla: DOC_TIPO1 12
Tabla: DOC_TIPO2 13
Tabla: DOC_TIPO3 13
Tabla: DOC_TIPO4 13
Tabla: EMPLEADOS 13
Tabla: IMPUESTOS 14
Tabla: MONEDA 14
Tabla: PAGOS 14

# CONTENIDO

Pág.


Tabla: PROVEEDORES 15
Tabla: USUARIOS 15

6. DICCIONARIO DE DATOS
FACTURA 16
ORDEN_DE_COMPRA 16
PROFORMA 17
NOTA_DE_CREDITO 17
NOTA_DE_DEBITO 17
EMISION_DE_CHEQUE 18
BOUCHER 18
COMPROBANTE_INGRESO_CAJA 19
COMPROBANTE_EGRESO_CAJA 20
LISTADO_DEL_INVENTARIO 20

7. ANALISIS DE CLASES
Aplicación Genérica 21
Aplicacion Gestión Comercial 24

8. DISEÑO DE CLASES

ESPECIFICACION DE CLASES
FUNCTIONAL CLASS: ClsSqlHandle 29
InitializeClass() 29
Initialize() 30
Error() 30
Connect() 30
SetIsolationLevel() 31
Prepare() 31
Execute() 32
Commit() 32
Disconnect() 32

FUNCTIONAL CLASS: ClsSqlHandleSelect
Execute() 33
First() / Last() 34
Previous() / Next() 34
FetchRow() 35

# CONTENIDO


| | | Pág. |
|---|---|---|
| DIALOG BOX CLASS: | ClsDlgReport | |
| View() | | 36 |
| Print() | | 37 |
| Initialize() | | 37 |
| GENERAL WINDOW CLASS: | ClsGenResetDirty | 38 |
| DATA FIELD CLASS: | ClsResetDirty | |
| FUNCTIONAL CLASS: | ClsSqlHandleSGC | |
| Error() | | 39 |
| SetParameter() | | 40 |
| GetResultSetCount() | | 40 |
| Error() | | 41 |
| FORM WINDOW CLASS: | Clsfrm_standardin | |
| OkToLoseChangeIfAny() | | 42 |
| BeginOperation() | | 42 |
| EndOperation() | | 43 |
| DisableClose() | | 43 |
| TABLE WINDOW CLASS: | Clstbl_standardout | |
| BeginOperation() | | 45 |
| EndOperation() | | 45 |
| DisableClose() | | 45 |

CONTENIDO

Pág.

# II. SQL Windows Solo Versión 5.0.0: Evaluación y Comparación de herramientas de DOO


9. CONCEPTOS BASICOS

| | | |
|---|---|---|
| 9.1 | Pilares de desarrollo de Aplicaciones Orientadas a Objetos | 47 |
| 9.1.1 | Clases | 47 |
| 9.1.2 | Clases SQLWindows | 47 |
| 9.1.3 | Encapsulamiento | 47 |
| 9.1.4 | Herencia | 48 |
| 9.1.5 | Polimorfismo | 48 |
| 9.2 | Beneficios de OOP | 49 |
| 9.3 | Diseño de Aplicaciones con OO | 49 |

10. DEFINICION DE CLASES

| | | |
|---|---|---|
| 10.1 | Tipos de Clases | 50 |
| 10.2 | Elementos de una Clase | 50 |
| 10.3 | Class Variables vs Instance Variables | 51 |

11. WINDOW CLASES

| | | |
|---|---|---|
| 11.1 | Top - Level Window Clases | 52 |
| 11.2 | Creando los Objetos | 53 |
| 11.3 | Ubicación en el Outline | 53 |
| 11.4 | Override de mensajes | 53 |

12. FUNCTIONAL CLASES

| | | |
|---|---|---|
| 12.1 | Usos | 54 |
| 12.2 | Variables definidas por el usuario | 54 |
| 12.3 | Functional class sin funciones | 55 |
| 12.4 | Functional class con funciones | 55 |
| | Funciones de la Clase | 55 |
| | LLamado de funciones | 55 |

# CONTENIDO


| | | Pág. |
|---|---|---|
| 13. | JERARQUIA DE CLASES | |
| 13.1 | Herencia Sencilla | 56 |
| 13.2 | Jerarquías | 56 |
| 13.3 | Herencia Múltiple | 57 |
| 13.4 | Derivando clases | 57 |
| 13.5 | Clasificación de referencias | |
| | Sin clasificación | 58 |
| | Clasificación por clase | 58 |
| | Clasificación completa de la clase | 58 |
| | Clasificación por objeto | 58 |
| | Clasificación completa del objeto | 58 |
| 13.6 | Reglas de derivación | 59 |
| 13.7 | General Window Class | 59 |
| 14. | POLIMORFISMO | |
| 14.1 | Binding | 60 |
| | Binding Estático | 60 |
| | Binding Dinámico | 60 |
| 15. | LA PROGRAMACION ESTRUCTURADA vs POO | 62 |
| 15.1 | OOP | 62 |
| 15.2 | Qué cosas pueden ser Objetos ? | 64 |
| 15.3 | Propiedades fundamentales de OOP | |
| | Abstracción de datos | 65 |
| | Encapsulamiento y ocultación de la información | 66 |
| | Herencia | 66 |
| | Polimorfismo | 68 |
| | Reutilización | 68 |
| 16. | SQLWINDOWS vs LENGUJES DE POO | |
| 16.1 | Smalltalk | 70 |
| 16.2 | Turbo/Quick/Objective Pascal | 71 |
| 16.3 | C++ | 71 |
| 16.4 | Turbo C++ y Borland C++ | 71 |
| 16.5 | Microsoft C++ 7.0 | 72 |
| 16.6 | Microsoft C/C++ 7.0 | 72 |
| 16.7 | SQLWindows | 73 |
| 16.7.1 | Un mejor método | 74 |
| 16.7.2 | Diseño de Clases frecuentemente usadas | 74 |
| 16.7.3 | Stándares Corporativos para interfaces de usuarios | 74 |
| 16.7.4 | Fácil mantenimiento de código | 75 |
| 16.7.5 | Ocultamiento de los detalles de la implementación | 75 |

# APENDICE

Copias de documentos comerciales negociables y no negociables

# 1. OBJETIVOS DEL PROYECTO

Tomamos como objetivo el desarrollo de una Aplicación que cubra Gestión Comercial y Gestión Contable, usando la Tecnología & Arquitectura Cliente / Servidor junto con el Paradigma del mundo Orientado a Objetos.

El énfasis más que a la parte aplicativa, se lo hemos dado en descubrir las ventajas, beneficios, desaveniencias, ... de estas 2 tendencias; de forma tal que, al culminar el proyecto, podamos sentirnos capaces de debatir, en un futuro, en Consultorías relacionadas a estos temas. Herramienta de desarrollo como diseñadores de Software nos interesa en cierta forma. Una que está creciendo en popularidad es, SQL Windows ( disponible en 4 versiones ) que fue de la mano con nuestro Tópico de Graduación y somos pioneros en usarla.

De lo mencionado anteriormente se subdividen miniobjetivos como encontrar:

- Qué tanto puede lograrse el 100% de reusabilidad ?
- Se puede eliminar la secuencialidad en la programación ?
- Diferencias con la Metodología Estructurada
- Descubrir formas de implementar los mensajes entre objetos
- Usando Metodología Orientada a Objetos que tanto me puedo abstraer de los detalles
- Fat Client, expresión más simple de Cliente / Servidor

.... entre muchos otros, que fueron apareciendo ...

# 2. DEFINICION DEL PROBLEMA

Desarrollar una Aplicación Cliente Servidor usando los conceptos de Análisis y Diseño Orientado a Objetos.

Tomamos como base el problema de Gestión Comercial en un esquema Multiempresarial. Este subsistema debe ofrecer datos para las operaciones típicas que se dan, como son:

- Gestión de Documentos:
  Facturas, Ordenes de Compra, Ordenes de Pedido, Notas de Entrega, Proformas,
  Ingreso / Egreso de bodega,
  Ingreso / Egreso de Caja,
  Registro de Bouchers, Emisión de Cheques,
  Letras de Cambio, Pagarés,
  Notas de Crédito / Débito
- Ventas: Análisis de Ventas, Listas de Precios, ...
- Compras: Análisis de Compras, Demanda de articulos, ...
- Manejo de Inventario: Reajuste de Inventario, Listado de Stock, Emisión de Backorder, Aplicación de Impuestos, ...
- Cuentas por Cobrar: Cartera, Cuentas vencidas, Cuentas por Vencer, ...
- Cuentas por Pagar: Cartera, Cuentas vencidas, Cuentas por Vencer, ...
  Directorio: datos de Proveedores, Clientes, Empleados, otros relacionados a la empresa
- Bancos: Conciliaciones bancarias, Registro de Depósitos, Operaciones (diarias, semanales, mensuales, ... ), ...

El subsistema de Gestión Contable que incluya:

- Plan de Cuentas definible
- Balances Diseñables: General, de Comprobación, Consolidado
- Estados Financieros
- Asientos de Ajustes
- Amortizaciones, Depreciaciones, ...
- Roles de Pago
- Libros Diarios, Mayores, Auxiliares, ...

Este último subsistema queda planteado como parte del problema para una versión futura.

# 2. DEFINICION DEL PROBLEMA

Desarrollar una Aplicación Cliente Servidor usando los conceptos de Análisis y Diseño Orientado a Objetos.

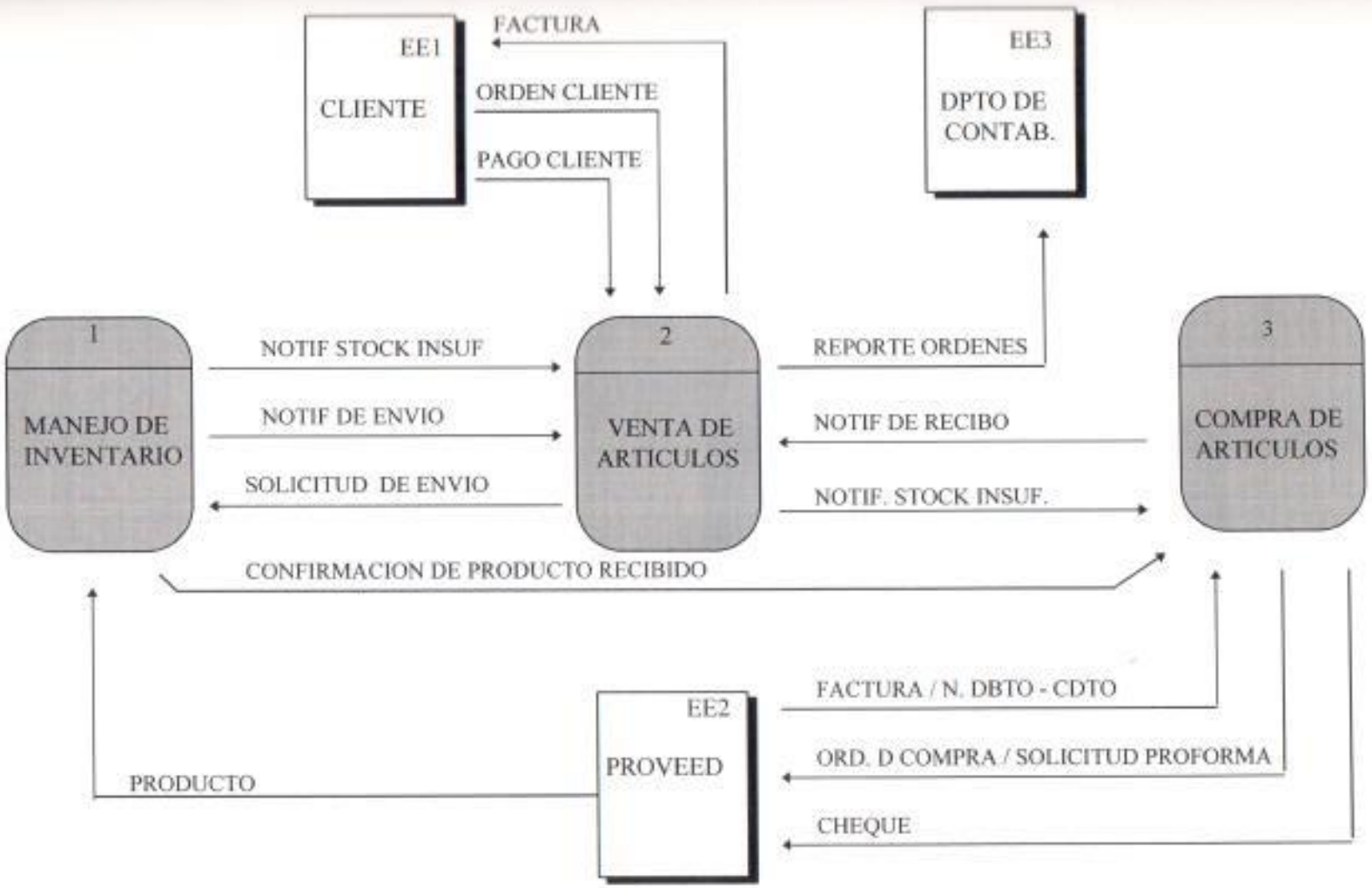
Tomamos como base el problema de Gestión Comercial en un esquema Multiempresarial. Este subsistema debe ofrecer datos para las operaciones típicas que se dan, como son:

- Gestión de Documentos:
  Facturas, Ordenes de Compra, Ordenes de Pedido, Notas de Entrega, Proformas,
  Ingreso / Egreso de bodega,
  Ingreso / Egreso de Caja,
  Registro de Bouchers, Emisión de Cheques,
  Letras de Cambio, Pagarés,
  Notas de Crédito / Débito
- Ventas: Análisis de Ventas, Listas de Precios, ...
- Compras: Análisis de Compras, Demanda de artículos, ...
- Manejo de Inventario: Reajuste de Inventario, Listado de Stock, Emisión de Backorder, Aplicación de Impuestos, ...
- Cuentas por Cobrar: Cartera, Cuentas vencidas, Cuentas por Vencer, ...
- Cuentas por Pagar: Cartera, Cuentas vencidas, Cuentas por Vencer, ...
  Directorio: datos de Proveedores, Clientes, Empleados, otros relacionados a la empresa
- Bancos: Conciliaciones bancarias, Registro de Depósitos, Operaciones (diarias, semanales, mensuales, ... ), ...

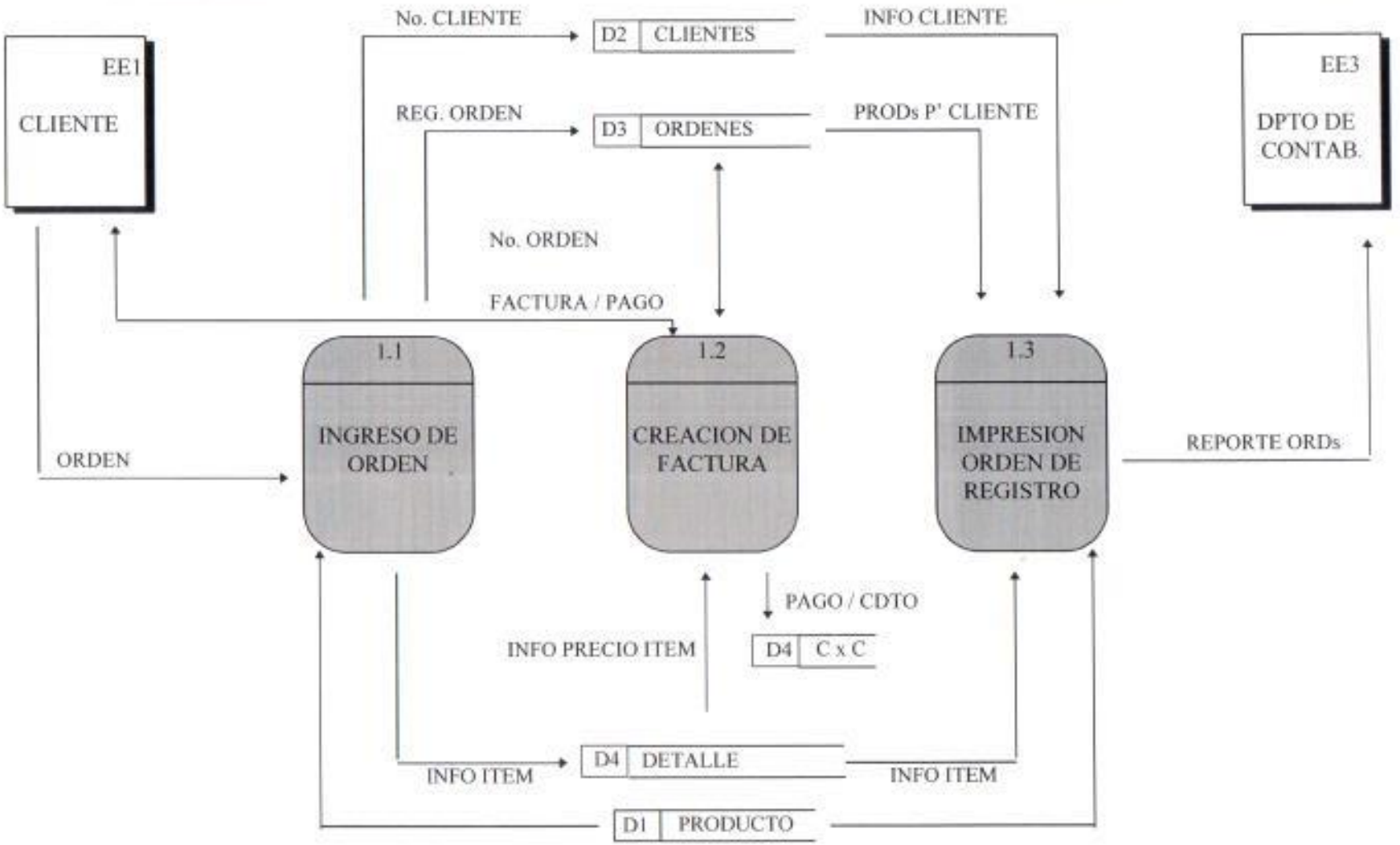
El subsistema de Gestión Contable que incluya:

- Plan de Cuentas definible
- Balances Diseñables: General, de Comprobación, Consolidado
- Estados Financieros
- Asientos de Ajustes
- Amortizaciones, Depreciaciones, ...
- Roles de Pago
- Libros Diarios, Mayores, Auxiliares, ...

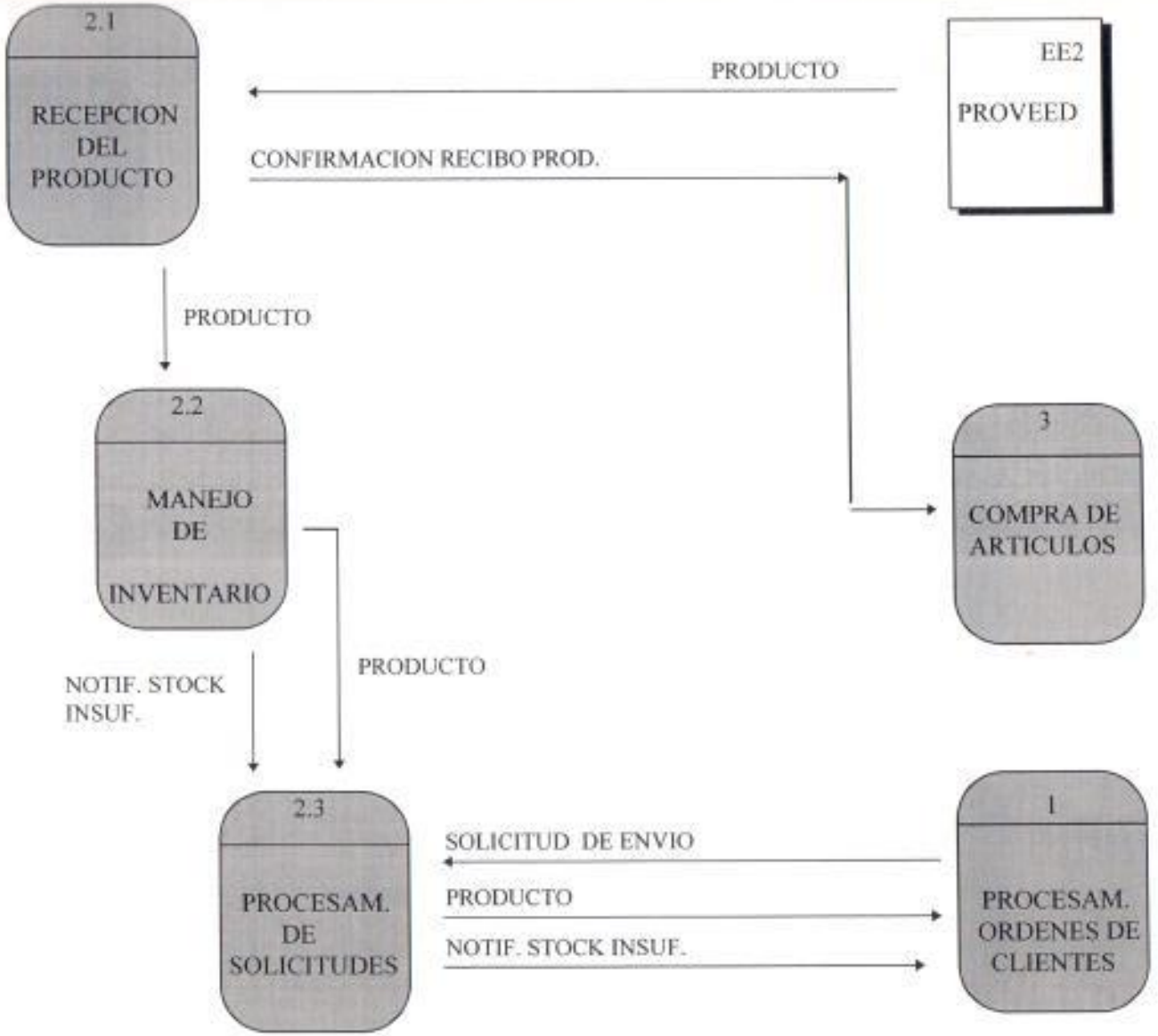
Este último subsistema queda planteado como parte del problema para una versión futura.


EE1
CLIENTE
FACTURA
ORDEN CLIENTE
PAGO CLIENTE
EE3
DPTO DE CONTAB.
1
MANEJO DE INVENTARIO
NOTIF STOCK INSUF
NOTIF DE ENVIO
SOLICITUD DE ENVIO
2
VENTA DE ARTICULOS
REPORTE ORDENES
NOTIF DE RECIBO
NOTIF. STOCK INSUF.
3
COMPRA DE ARTICULOS
CONFIRMACION DE PRODUCTO RECIBIDO
EE2
PROVEED
PRODUCTO
FACTURA / N. DBTO - CDTO
ORD. D COMPRA / SOLICITUD PROFORMA
CHEQUE

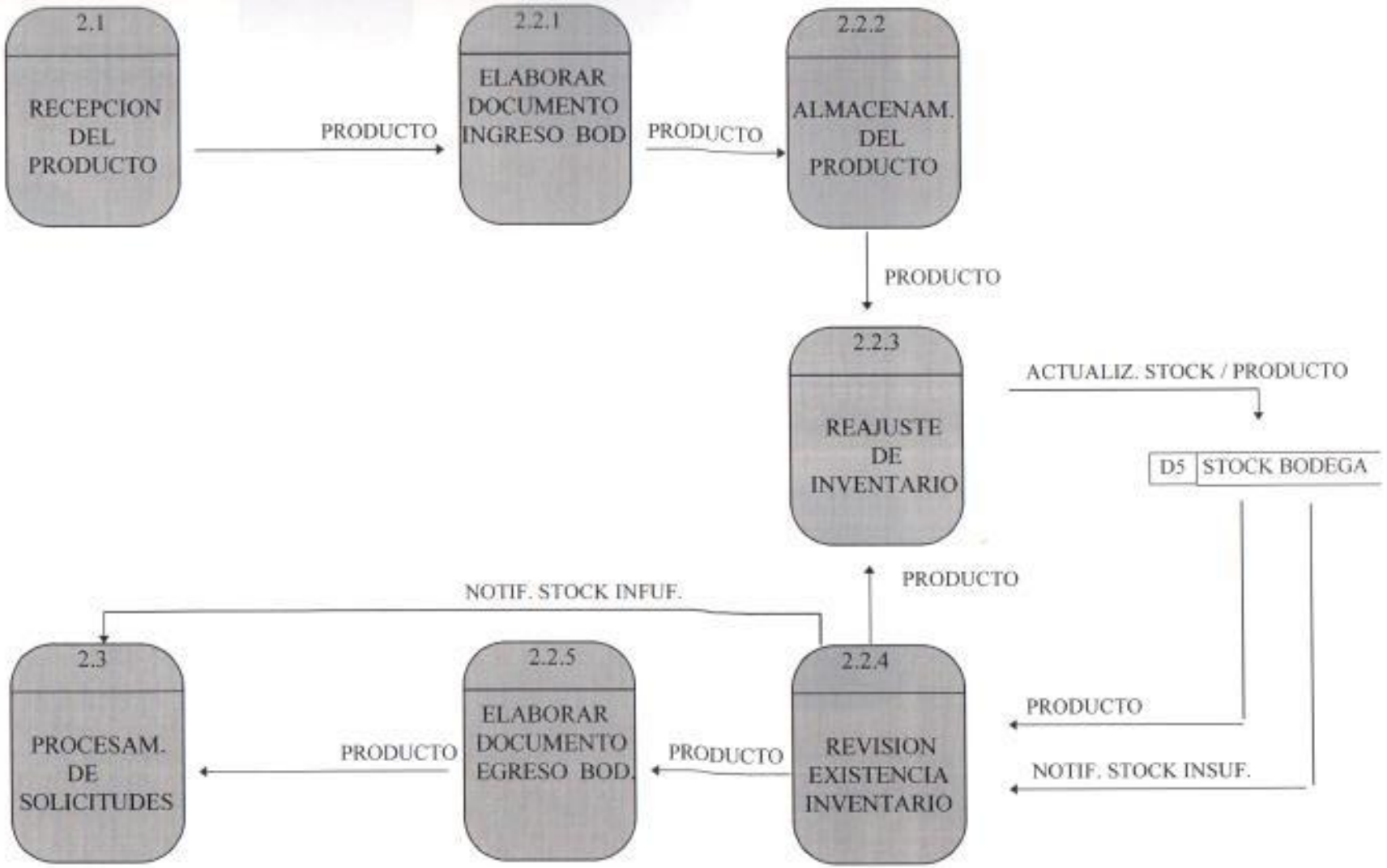
# 1. VENTA DE ARTICULOS

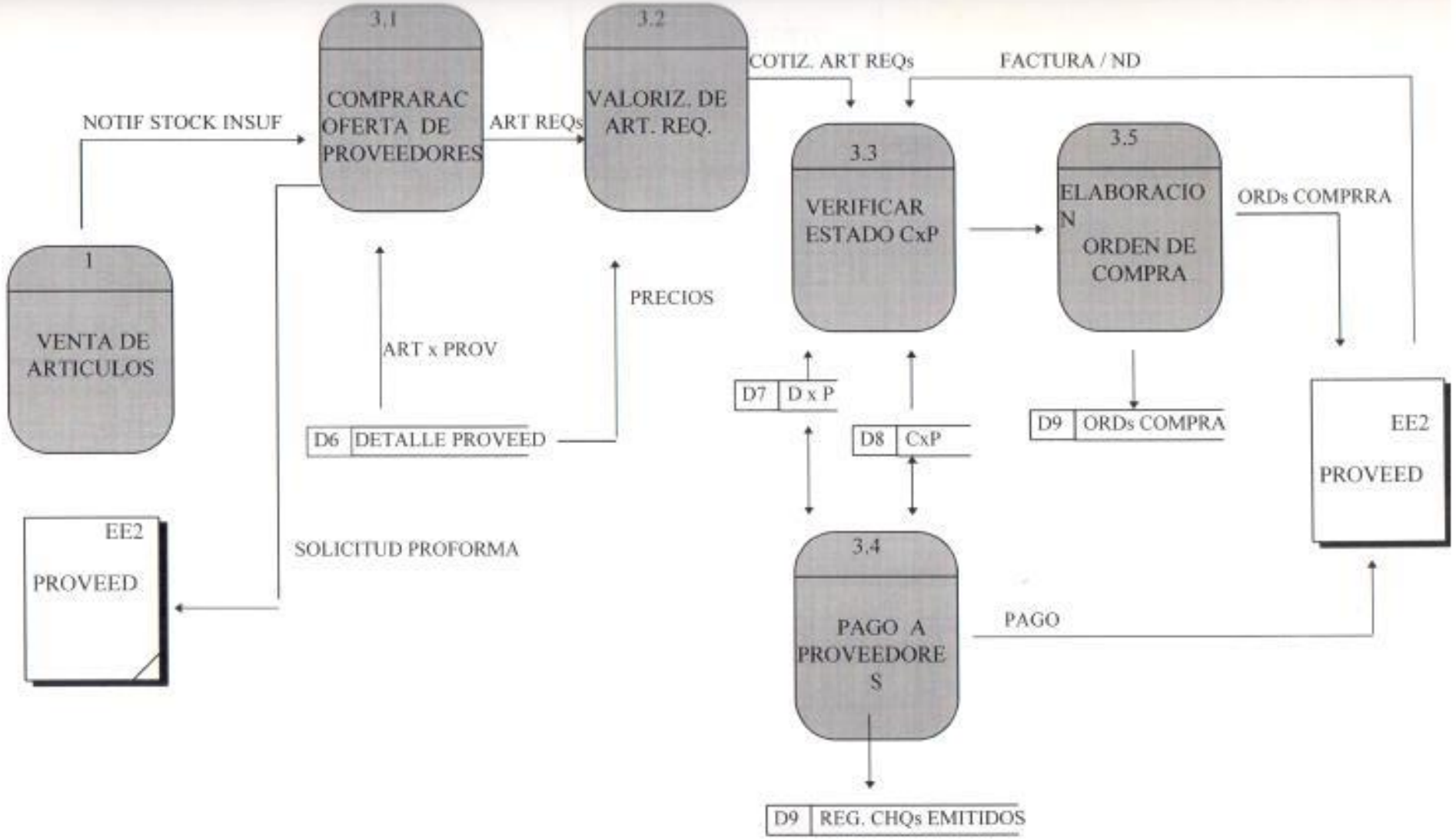


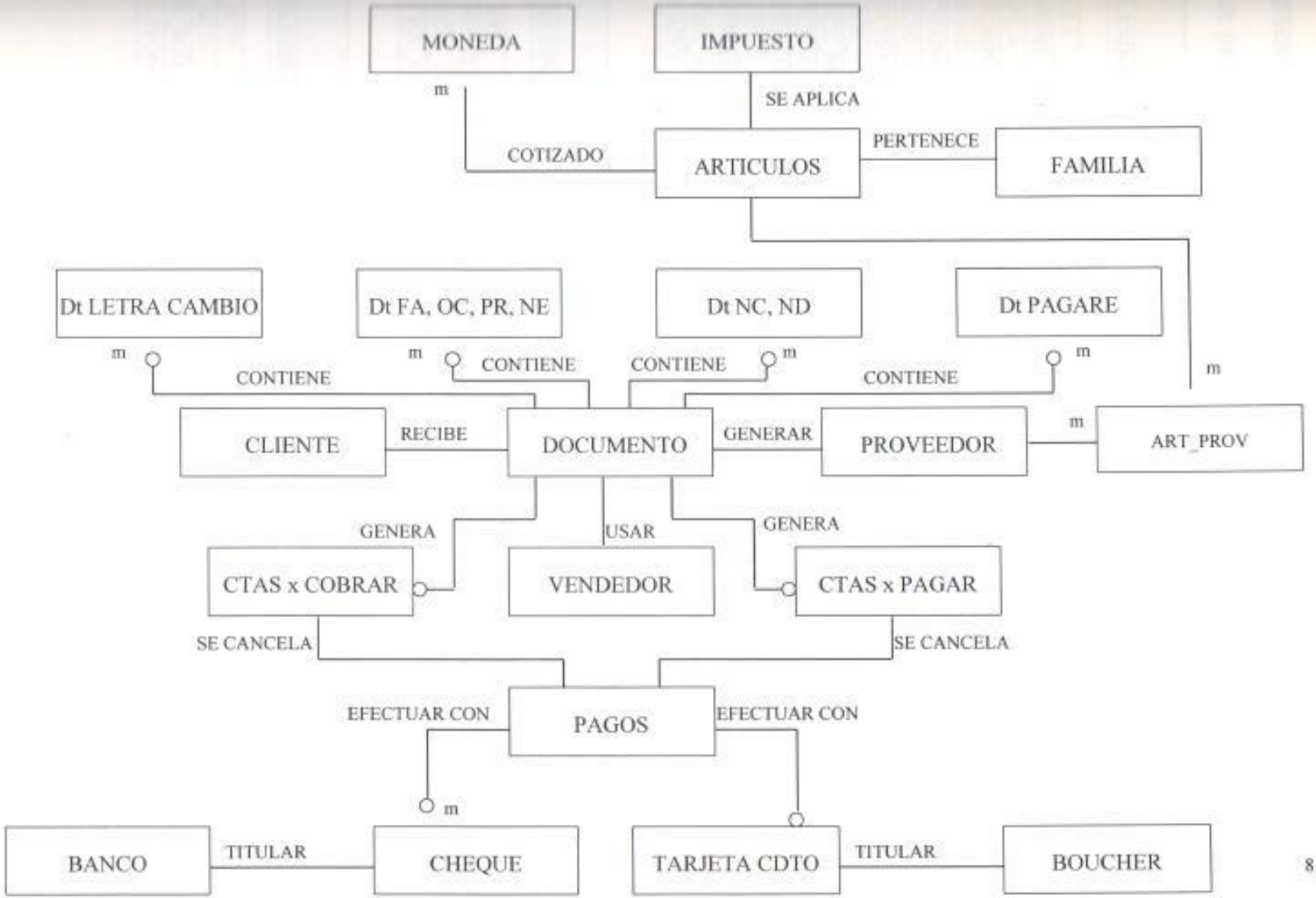
# MANEJO DE INVENTARIO



## 2.2 MANEJO DE INVENTARIO




3.1
COMPRARAC OFERTA DE PROVEEDORES
3.2
VALORIZ. DE ART. REQ.
COTIZ. ART REQs
FACTURA / ND
NOTIF STOCK INSUF
ART REQs
3.3
VERIFICAR ESTADO CxP
3.5
ELABORACIO N ORDEN DE COMPRA
ORDs COMPRRA
1
VENTA DE ARTICULOS
ART x PROV
PRECIOS
D7 D x P
D8 CxP
D9 ORDs COMPRA
EE2
PROVEED
D6 DETALLE PROVEED
EE2
PROVEED
SOLICITUD PROFORMA
3.4
PAGO A PROVEEDORE S
PAGO
D9 REG. CHQs EMITIDOS


MONEDA
IMPUESTO
m
SE APLICA
COTIZADO
ARTICULOS
PERTENECE
FAMILIA
Dt LETRA CAMBIO
Dt FA, OC, PR, NE
Dt NC, ND
Dt PAGARE
m
CONTIENE
CONTIENE
CONTIENE
CONTIENE
m
CLIENTE
RECIBE
DOCUMENTO
GENERAR
PROVEEDOR
ART_PROV
GENERA
USAR
GENERA
CTAS x COBRAR
VENDEDOR
CTAS x PAGAR
SE CANCELA
SE CANCELA
EFECTUAR CON
PAGOS
EFECTUAR CON
BANCO
TITULAR
CHEQUE
TARJETA CDTO
TITULAR
BOUCHER

## 5. DISEÑO ESTRUCTURADO DE DATOS: Modelo Fisico de Datos

### BASE DE DATOS: GESCOM

**TABLA: ARTICULOS**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| C_ART | CHAR | 4 | N |
| NOMBRE_ART | CHAR | 20 | N |
| MARCA | CHAR | 30 | N |
| DESCRIPCION | CHAR | 10 | N |
| UNIDAD | CHAR | 5 | N |
| FECHAINGRESO | DATE | | N |
| PRECIO_VENTA | DECIMAL | 7,2 | N |
| DESCUENTO | DECIMAL | 7,2 | Y |
| STOCK_MAX | INTEGER | | N |
| STOCK_MIN | INTEGER | | N |
| STOCK_ACTUAL | INTEGER | | N |
| FECHA_ACTUALIZ | DATE | | N |
| MOTIVO_AJUSTE | CHAR | 2 | Y |
| CANTIDAD_AJUSTE | INTEGER | | Y |
| FECHA_AJUSTE | DATE | | Y |
| ESTADO | CHAR | 2 | N |

**TABLA: ART_PROV**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| C_ART | CHAR | 4 | N |
| C_PROVEEDOR | CHAR | 4 | N |
| PRECIO_COSTO | DECIMAL | 7,2 | N |
| FECHA_ACTCOSTO | DATE | | N |

**TABLA: BANCOS**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| C_BCO | CHAR | 4 | N |
| N_BCO | CHAR | 20 | N |
| DIR_BCO | CHAR | 30 | Y |
| TELEFONO | CHAR | 8 | N |

**TABLA: FAMILIA**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| C_FAMILIA | CHAR | 4 | N |
| NOMBRE | CHAR | 20 | N |
| DESCRIPCION | CHAR | 20 | Y |

**TABLA: CLIENTES**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| C_CLIENTE | CHAR | 4 | N |
| NOMBRE | CHAR | 30 | N |
| TIPO_CLIENTE | CHAR | 1 | N |
| CDTO_MAX | DECIMAL | 5,2 | Y |
| TELEFONO1 | CHAR | 8 | N |
| TELEFONO2 | CHAR | 8 | Y |
| TELEFONO3 | CHAR | 8 | Y |
| FAX1 | CHAR | 8 | Y |
| FAX2 | CHAR | 8 | Y |
| FAX3 | CHAR | 8 | Y |
| C_POSTAL1 | CHAR | 5 | Y |
| C_POSTAL2 | CHAR | 5 | Y |
| C_POSTAL3 | CHAR | 5 | Y |
| REP_COMPRAS1 | CHAR | 20 | N |
| REP_COMPRAS2 | CHAR | 20 | Y |
| REP_COMPRAS3 | CHAR | 20 | Y |
| CARGO1 | CHAR | 15 | Y |
| CARGO2 | CHAR | 15 | Y |
| CARGO3 | CAHR | 15 | Y |
| CIUDAD1 | CHAR | 10 | Y |
| CIUDAD2 | CHAR | 10 | Y |
| CIUDAD3 | CHAR | 10 | Y |
| DIRECCION1 | CHAR | 35 | N |
| DIRECCION2 | CHAR | 35 | Y |
| DIRECCION3 | CHAR | 35 | Y |

**TABLA: CTA_CTE**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| C_CTACTE | CHAR | 4 | N |
| C_DUEÑO | CHAR | 4 | N |
| C_BANCO | CHAR | 4 | N |

| TABLA: CUENTAS | | | |
|---|---|---|---|
| **COLUMNA** | **TIPO** | **LONGITUD** | **NULOS** |
| | | | |
| C_DOCUM | CHAR | 4 | N |
| FECHA_EMISION | DATE | | N |
| FECHA_VENCE | DATE | | N |
| NU_PAGOS | INTEGER | | Y |
| VALOR | DECIMAL | 7,2 | N |
| SALDO | DECIMAL | 7,2 | N |
| C_INTERES | CHAR | 4 | Y |
| C_MORA | CHAR | 4 | Y |

| TABLA: DETALLE_ORDEN | | | |
|---|---|---|---|
| **COLUMNA** | **TIPO** | **LONGITUD** | **NULOS** |
| | | | |
| C_DOCUM | CHAR | 4 | |
| C_ART | CHAR | 4 | |
| DESCUENTO | DECIMAL | 7,2 | |
| PRECIO_VENTA | DECIMAL | 7,2 | |
| CANTIDAD | INTEGER | | |
| CONCEPTO | CHAR | 20 | |
| TIPO_DOC | CHAR | 2 | |

| TABLA: DOCUMENTOS | | | |
|---|---|---|---|
| **COLUMNA** | **TIPO** | **LONGITUD** | **NULOS** |
| | | | |
| C_DOCUM | CHAR | 4 | N |
| C_REFENCIA | CHAR | 4 | N |
| FECHA_EMISION | DATE | | N |
| VALOR | DECIMAL | 7,2 | N |
| ESTADO | CHAR | 2 | N |
| TIPO_DOC | CHAR | 2 | N |
| VALOR_PALABRAS | CHAR | 30 | Y |
| C_MONEDA | CHAR | 2 | Y |

**TABLA: CLIENTES**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| | | | |
| C_CLIENTE | CHAR | 4 | N |
| NOMBRE | CHAR | 30 | N |
| TIPO_CLIENTE | CHAR | 1 | N |
| CDTO_MAX | DECIMAL | 5,2 | Y |
| TELEFONO1 | CHAR | 8 | N |
| TELEFONO2 | CHAR | 8 | Y |
| TELEFONO3 | CHAR | 8 | Y |
| FAX1 | CHAR | 8 | Y |
| FAX2 | CHAR | 8 | Y |
| FAX3 | CHAR | 8 | Y |
| C_POSTAL1 | CHAR | 5 | Y |
| C_POSTAL2 | CHAR | 5 | Y |
| C_POSTAL3 | CHAR | 5 | Y |
| REP_COMPRAS1 | CHAR | 20 | N |
| REP_COMPRAS2 | CHAR | 20 | Y |
| REP_COMPRAS3 | CHAR | 20 | Y |
| CARGO1 | CHAR | 15 | Y |
| CARGO2 | CHAR | 15 | Y |
| CARGO3 | CHAR | 15 | Y |
| CIUDAD1 | CHAR | 10 | Y |
| CIUDAD2 | CHAR | 10 | Y |
| CIUDAD3 | CHAR | 10 | Y |
| DIRECCION1 | CHAR | 35 | N |
| DIRECCION2 | CHAR | 35 | Y |
| DIRECCION3 | CHAR | 35 | Y |

**TABLA: DOC_TIPO1**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| | | | |
| C_DOCUM | CHAR | 4 | N |
| CLIENTE_PROV | CHAR | 1 | N |
| C_CLIPRO | CHAR | 4 | N |
| SALDO | DECIMAL | 7,2 | N |
| ABONO | DECIMAL | 7,2 | N |
| CTDO_CTDO | CHAR | 1 | N |
| FECHA_VENCE | DATE | | N |
| C_OC | CHAR | 4 | Y |
| C_NE | CHAR | 4 | Y |
| SUBTOTAL | DECIMAL | 7,2 | N |

**TABLA: DOC_TIPO2**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| | | | |
| C_DOCUM | CHAR | 4 | N |
| NC_ND | CHAR | 1 | N |
| DETALLE | CHAR | 40 | N |
| VALOR_DETALLE | DECIMAL | 7,2 | N |
| C_PERSONA | CHAR | 4 | N |

**TABLA: DOC_TIPO3**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| | | | |
| C_DOCUM | CHAR | 4 | N |
| C_CTACTE | CHAR | 4 | Y |
| NUM_CHQ | INTEGER | | N |
| C_PERSONA | CHAR | 4 | N |

**TABLA: DOC_TIPO4**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| | | | |
| C_DOCUM | CHAR | 4 | N |
| C_GARANTE | CHAR | 4 | Y |
| C_DEUDOR | CHAR | 4 | N |
| C_ENDOSANTE | CHAR | 4 | N |
| INTERES | CHAR | 2 | N |
| FECHA_VENCE | DATE | | N |

**TABLA: EMPLEADOS**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| | | | |
| C_EMPLEADO | CHAR | 4 | N |
| NOMBRES | CHAR | 15 | Y |
| APELLIDOS | CHAR | 15 | N |
| CARGO | CHAR | 15 | N |
| DIRECCION | CHAR | 35 | N |
| TELEFONO | CHAR | 8 | N |
| FECHA_NAC | DATE | | |
| CIUDAD_NAC | CHAR | 15 | N |
| C_JEFE | CHAR | 4 | N |
| STATUS | CHAR | 2 | Y |

| TABLA: IMPUESTOS | | | |
|---|---|---|---|
| **COLUMNA** | **TIPO** | **LONGITUD** | **NULOS** |
| | | | |
| C_IMPUESTOS | CHAR | 2 | N |
| N_IMPUESTOS | CHAR | 40 | Y |
| VALOR | CHAR | 2 | N |
| FECHA_DESDE | DATE | | N |
| FECHA_HASTA | CHAR | 35 | N |

| TABLA: MONEDA | | | |
|---|---|---|---|
| **COLUMNA** | **TIPO** | **LONGITUD** | **NULOS** |
| | | | |
| C_MONEDA | CHAR | 4 | N |
| MONEDA | CHAR | 30 | N |
| VALOR_SUCRES | DECIMAL | 7,2 | N |
| FECHA_DESDE | DATE | | N |
| FECHA_HASTA | DATE | | N |

| TABLA: PAGOS | | | |
|---|---|---|---|
| **COLUMNA** | **TIPO** | **LONGITUD** | **NULOS** |
| | | | |
| C_PAGO | CHAR | 4 | N |
| C_DOCUMENTO | CHAR | 15 | N |
| FECHA | DATE | | N |
| VALOR | DECIMAL | 7,2 | N |
| TIPO_DOC | CHAR | 2 | N |
| LET_PAG | CHAR | 1 | N |
| CHQ_TAR | CHAR | 1 | N |
| C_CHQBOU | CHAR | 4 | Y |
| C_BCOTAR | CHAR | 4 | Y |

**TABLA: PROVEEDORES**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| | | | |
| C_PROVEEDOR | CHAR | 4 | N |
| NOMBRE | CHAR | 30 | N |
| RUC | CHAR | 15 | N |
| TIPO_PROV | CHAR | 1 | N |
| CDTO_MAX | DECIMAL | 5,2 | Y |
| TELEFONO1 | CHAR | 8 | N |
| TELEFONO2 | CHAR | 8 | Y |
| TELEFONO3 | CHAR | 8 | Y |
| FAX1 | CHAR | 8 | Y |
| FAX2 | CHAR | 8 | Y |
| FAX3 | CHAR | 8 | Y |
| C_POSTAL1 | CHAR | 5 | Y |
| C_POSTAL2 | CHAR | 5 | Y |
| C_POSTAL3 | CHAR | 5 | Y |
| REP_VENTAS1 | CHAR | 20 | N |
| REP_VENTAS2 | CHAR | 20 | Y |
| REP_VENTAS3 | CHAR | 20 | Y |
| CARGO1 | CHAR | 15 | Y |
| CARGO2 | CHAR | 15 | Y |
| CARGO3 | CAHR | 15 | Y |
| CIUDAD1 | CHAR | 10 | Y |
| CIUDAD2 | CHAR | 10 | Y |
| CIUDAD3 | CHAR | 10 | Y |
| DIRECCION1 | CHAR | 35 | N |
| DIRECCION2 | CHAR | 35 | Y |
| DIRECCION3 | CHAR | 35 | Y |

**TABLA: USUARIOS**

| COLUMNA | TIPO | LONGITUD | NULOS |
|---|---|---|---|
| | | | |
| C_USUARIO | CHAR | 4 | N |
| ALIAS | CHAR | 10 | N |
| CONTRASENA | CHAR | 8 | N |
| M_VENTAS | CHAR | 1 | N |
| M_COMPRAS | CHAR | 1 | N |
| M_CUENTAS | CHAR | 1 | N |
| M_INVENTARIO | CHAR | 1 | N |

## 6. DICCIONARIO DE DATOS

| | | |
|---|---|---|
| **FACTURA** | = | Encabezado_de_factura |
| | + | { Items_de _factura } |
| | + | Pie_de_factura |
| Encabezado_de_factura | = | Factura_No |
| | + | Fecha_emisión |
| | + | Cuenta_No |
| | + | Nombre_cliente |
| | + | Dirección_cliente |
| | + | Orden_cliente_No |
| | + | Orden_Interna_No |
| Items_de_factura | = | Nombre_producto |
| | + | Código_producto |
| | + | Cantidad_pedida |
| | + | Precio_unitario |
| | + | Valor |
| Pie_de_factura | = | Subtotal |
| | + | IVA |
| | + | ( Descuento ) |
| | + | Total_a_Pagar |
| **ORDEN_DE_COMPRA** | = | Encabezado_de_orden |
| | + | { Items_de _Orden } |
| | + | Pie_Orden |
| Encabezado_de_Orden | = | Orden_No |
| | + | Fecha_emisión |
| | + | Dirigido_a |
| | + | Condiciones_de_pago |
| | + | Tiempo_de_entrega |
| Items_de_factura | = | Item_No |
| | + | Descripción |
| | + | Cantidad |
| | + | Precio_unitario |
| | + | Valor |
| Pie_de_orden | = | Valor_total |
| | + | Solicitado_por |
| | + | Aprobado_por |

| | | |
|---|---|---|
| **PROFORMA** | = | Encabezado_de_Proforma |
| | + | {Items_de _Proforma} |
| | + | Autorizado_por |
| Encabezado_de_Proforma | = | Proforma_No |
| | + | Fecha_emisión |
| | + | Dirigido_a |
| | + | Dirección_cliente |
| | + | Condiciones_pago |
| | + | Tiempo_validez |
| | + | Tiempo_de_entrega |
| Items_de_Proforma | = | Item_No |
| | + | Descripción |
| | + | Cantidad |
| | + | Valor_unitario |
| | + | Valor |
| **NOTA_DE_CREDITO** | = | Encabezado_de_NotaCrédito |
| | + | {Items_de _NotaCrédito} |
| | + | Autorizado_por |
| Encabezado_de_NotaCrédito | = | Por_valor_de |
| | + | A_favor_de |
| | + | Dirección |
| | + | Fecha |
| Items_de_NotaCrédito | = | Concepto |
| | + | Valor |
| **NOTA_DE_DEBITO** | = | Encabezado_de_NotaDébito |
| | + | {Items_de _NotaDébito} |
| | + | Autorizado_por |
| Encabezado_de_NotaDébito | = | Por_valor_de |
| | + | A_cargo_de |
| | + | Dirección |
| | + | Fecha |
| Items_de_NotaDébito | = | Concepto |
| | + | Valor |

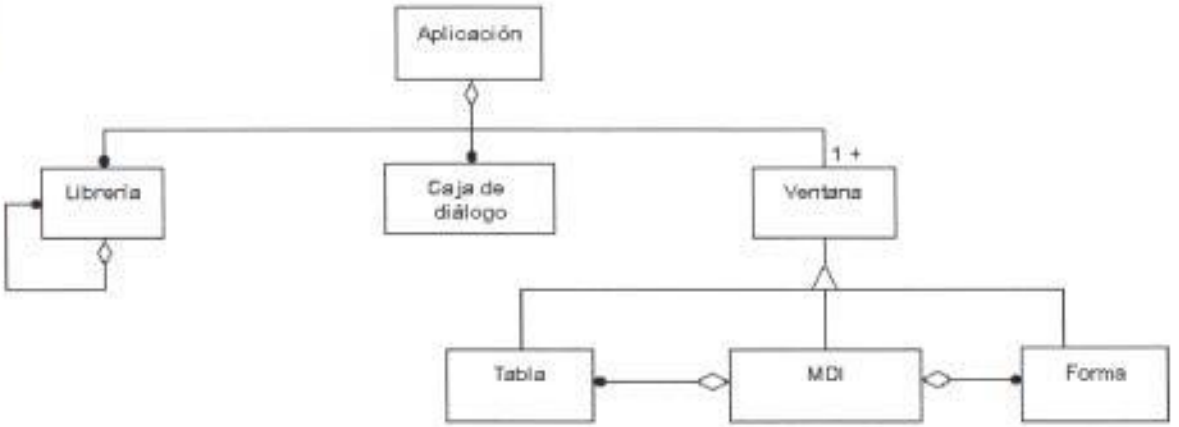
| | | |
|---|---|---|
| **EMSION_DE_CHEQUE** | = | Banco |
| | + | Nu_CtaCte |
| | + | Nu_Cheque |
| | + | Fecha |
| | + | Valor_Palabras |
| | + | { Items_emisión } |
| | + | Pie_emisión |
| Items_emisión | = | Concepto |
| | | Código |
| | | [ Valor_Debe \| Valor_Haber ] |
| Pie_emisión | = | Preparado_por |
| | | Revisado_por |
| | | Aprobado_por |
| | | ( Observaciones ) |
| | | CI_RUC |
| **BOUCHER** | = | Código |
| | + | Día |
| | + | Mes |
| | + | Año |
| | + | Nombre_cliente |
| | + | Valor_consumos_compras |
| | + | (Valor_impuesto) |
| | + | (Valor_propinas) |
| | + | Subtotal_impuestos_propinas |
| | + | Total_a_cargo |

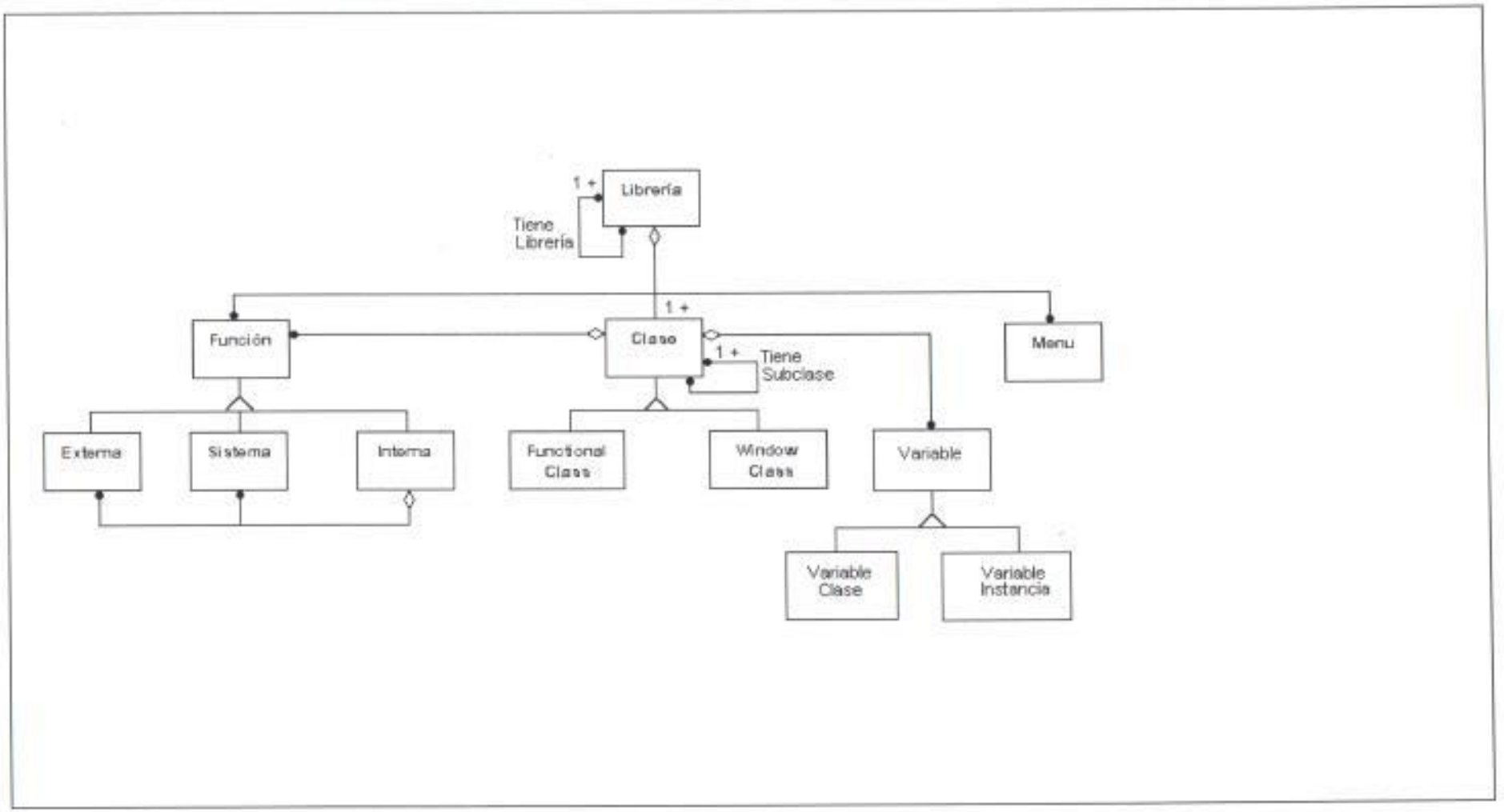
| | | |
|---|---|---|
| **COMPROBANTE_INGRESO_CAJA** | = | Encabezado |
| | + | { Detalle_comprobante } |
| | + | ( Total_Cheques ) |
| | + | ( Total_Efectivo ) |
| | + | Valor_total |
| | + | { Detalle_movimiento } |
| | + | Nombre_Cajera |
| | + | Nombre_quien_revisa |
| Encabezado | = | Nu_cmpbte |
| | + | Fecha |
| | + | Ciudad |
| | + | Recibimos_de |
| | + | Valor_palabras |
| | + | Concepto_del_ingreso |
| Detalle_comprobante | = | Nu_Factura |
| | + | ( Nu_Cheque ) |
| | + | ( Banco ) |
| | + | Valor |
| Detalle_movimiento | = | Código |
| | + | [ Valor_debe \| Valor_haber ] |

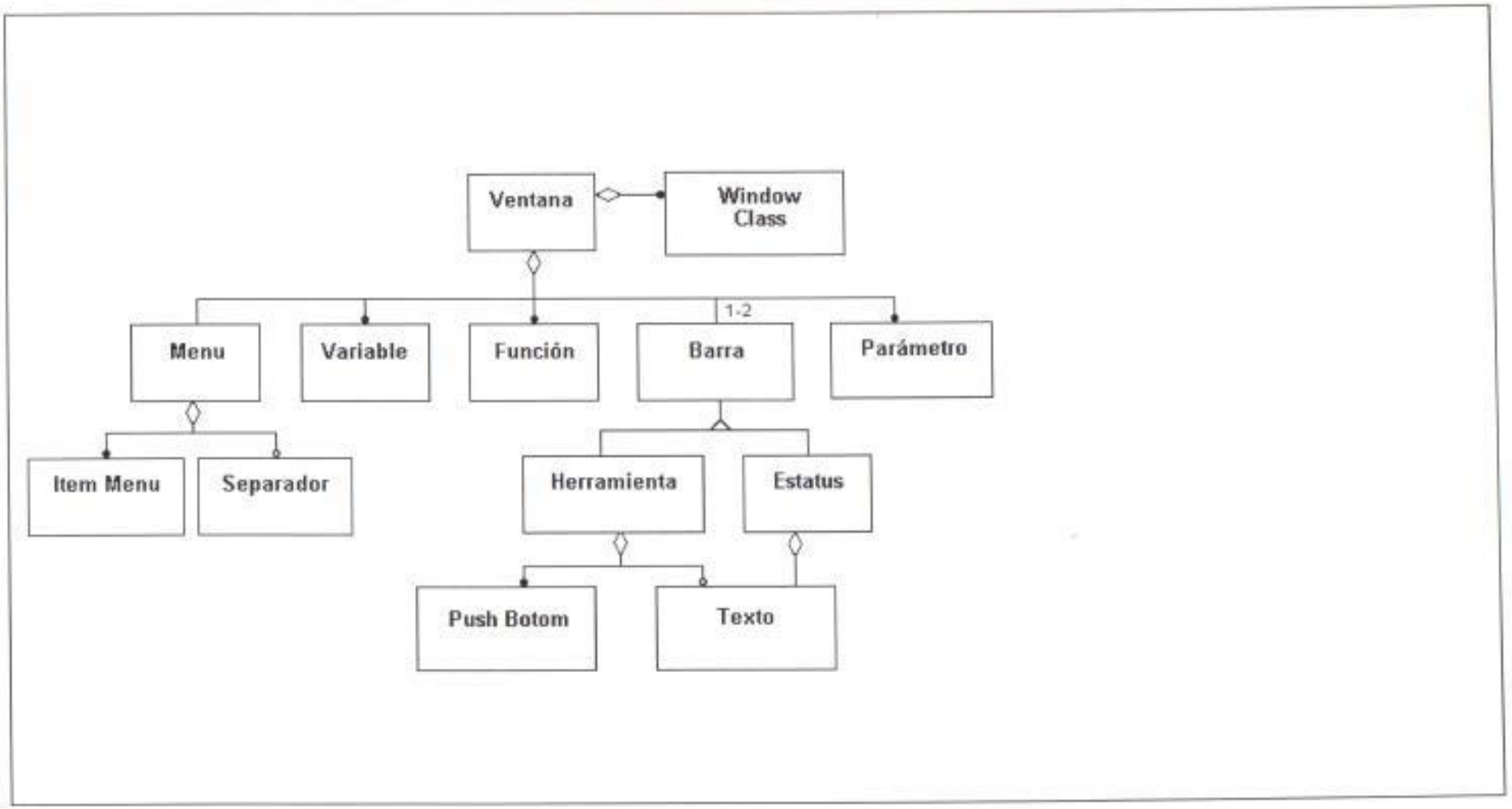
| | | |
|---|---|---|
| **COMPROBANTE_EGRESO_CAJA** | = | Encabezado |
| | + | { Detalle_comprobante } |
| | + | ( Total_Cheques ) |
| | + | ( Total_Efectivo ) |
| | + | Valor_total |
| | + | { Detalle_movimiento } |
| | + | Nombre_Cajera |
| | + | Nombre_quien_autoriza |
| Encabezado | = | Nu_cmpbte |
| | + | Fecha |
| | + | Ciudad |
| | + | Pagar_a |
| | + | Valor_palabras |
| | + | Concepto_del_egreso |
| Detalle_comprobante | = | Nu_Factura |
| | + | ( Nu_Cheque ) |
| | + | ( Banco ) |
| | + | Valor |
| Detalle_movimiento | = | Código |
| | + | [ Valor_debe \| Valor_haber ] |

| | | |
|---|---|---|
| **LISTADO DEL INVENTARIO** | = | Código_Producto |
| | + | Descripción_del_Producto |
| | + | Código_del_Producto |
| | + | Nombre_del_Proveedor |
| | + | { Nu_Bodega |
| | + | Nombre_Bodega |
| | + | Cantidad |
| | + | Stock_Mínimo } |

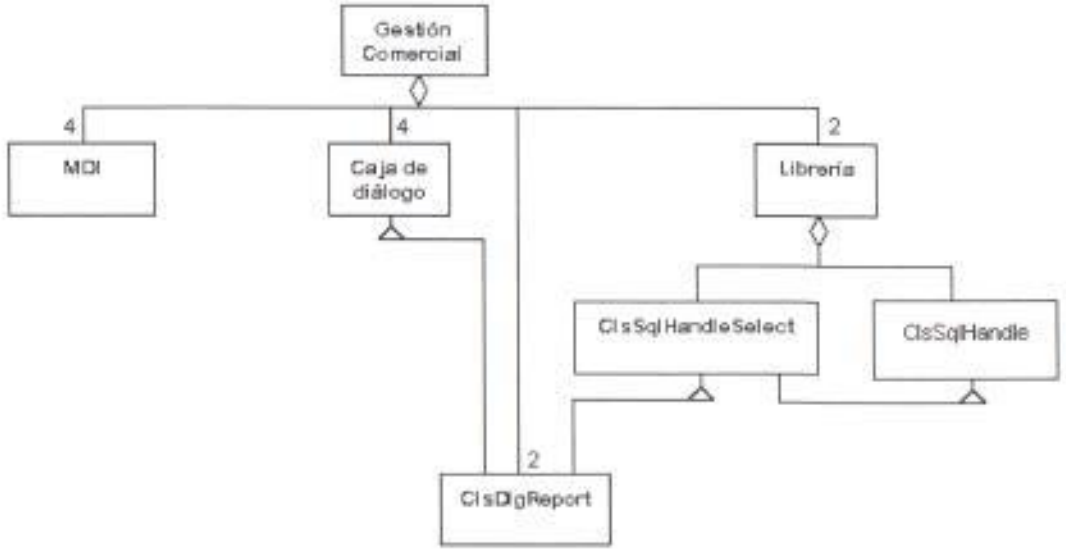
# 7. ANALISIS DE CLASES: APLICACION GENERICA

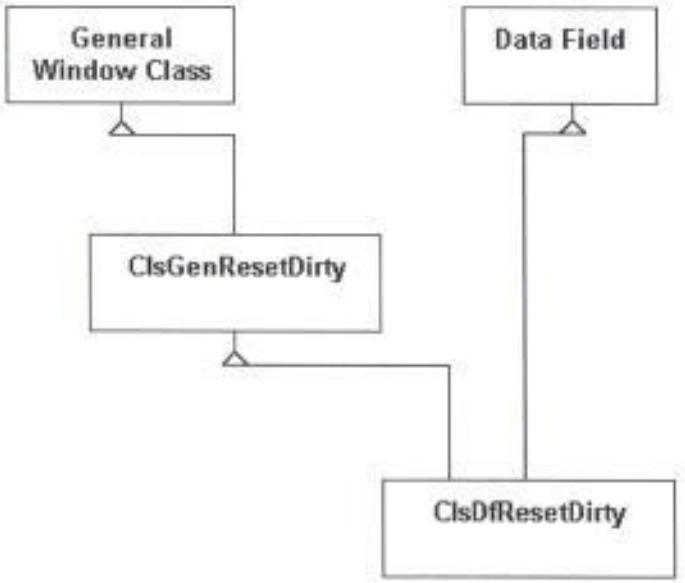



1 +
Librería
Tiene
Librería
1 +
Función
Clase
1 +
Tiene
Subclase
Menu
Externa
Sistema
Interna
Functional
Class
Window
Class
Variable
Variable
Clase
Variable
Instancia


Ventana
Window Class
Menu
Variable
Función
1-2
Barra
Parámetro
Item Menu
Separador
Herramienta
Estatus
Push Botom
Texto

## APLICACION: GESTION COMERCIAL




General
Window Class
Data Field
ClsGenResetDirty
ClsDfResetDirty

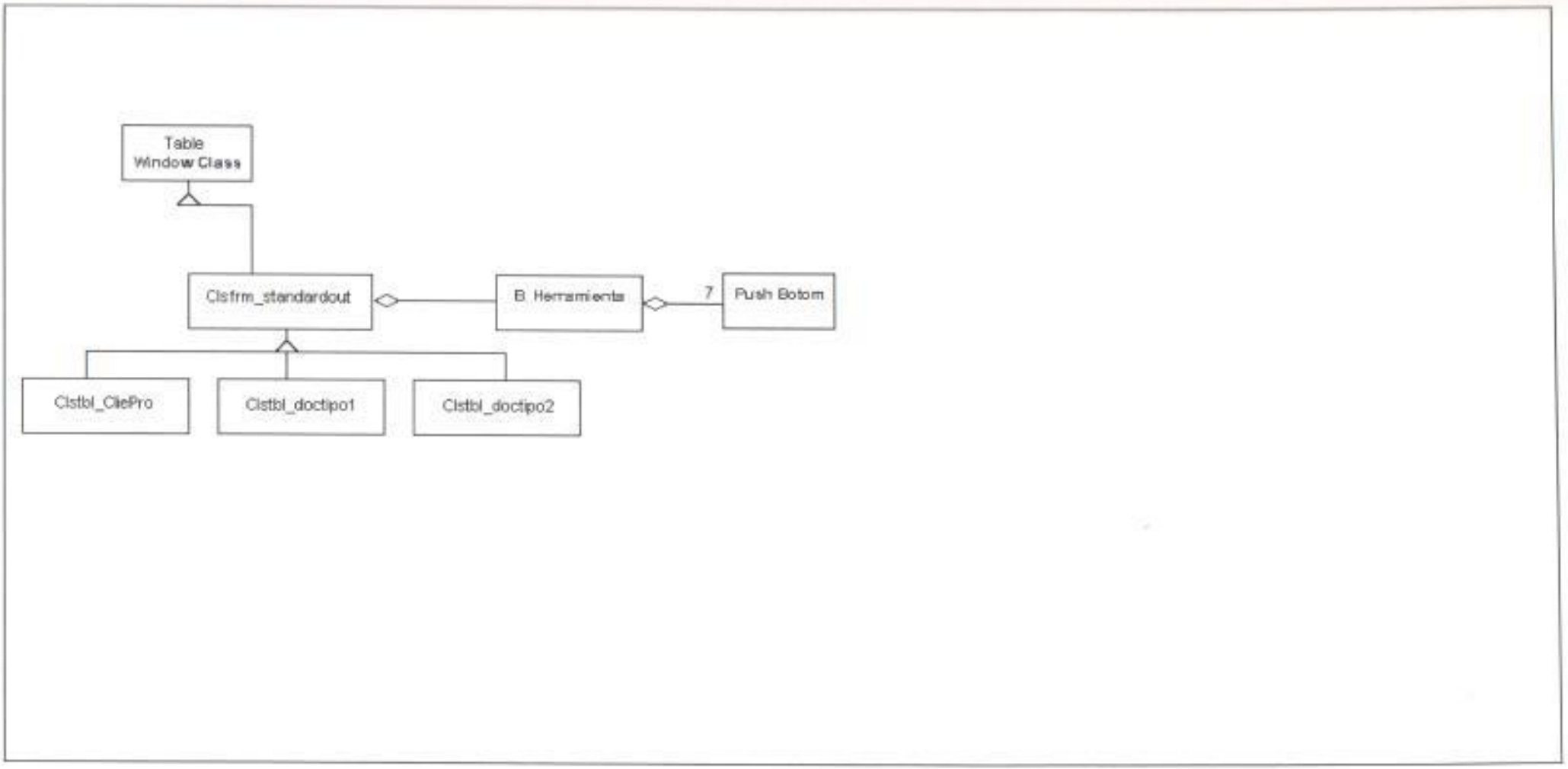
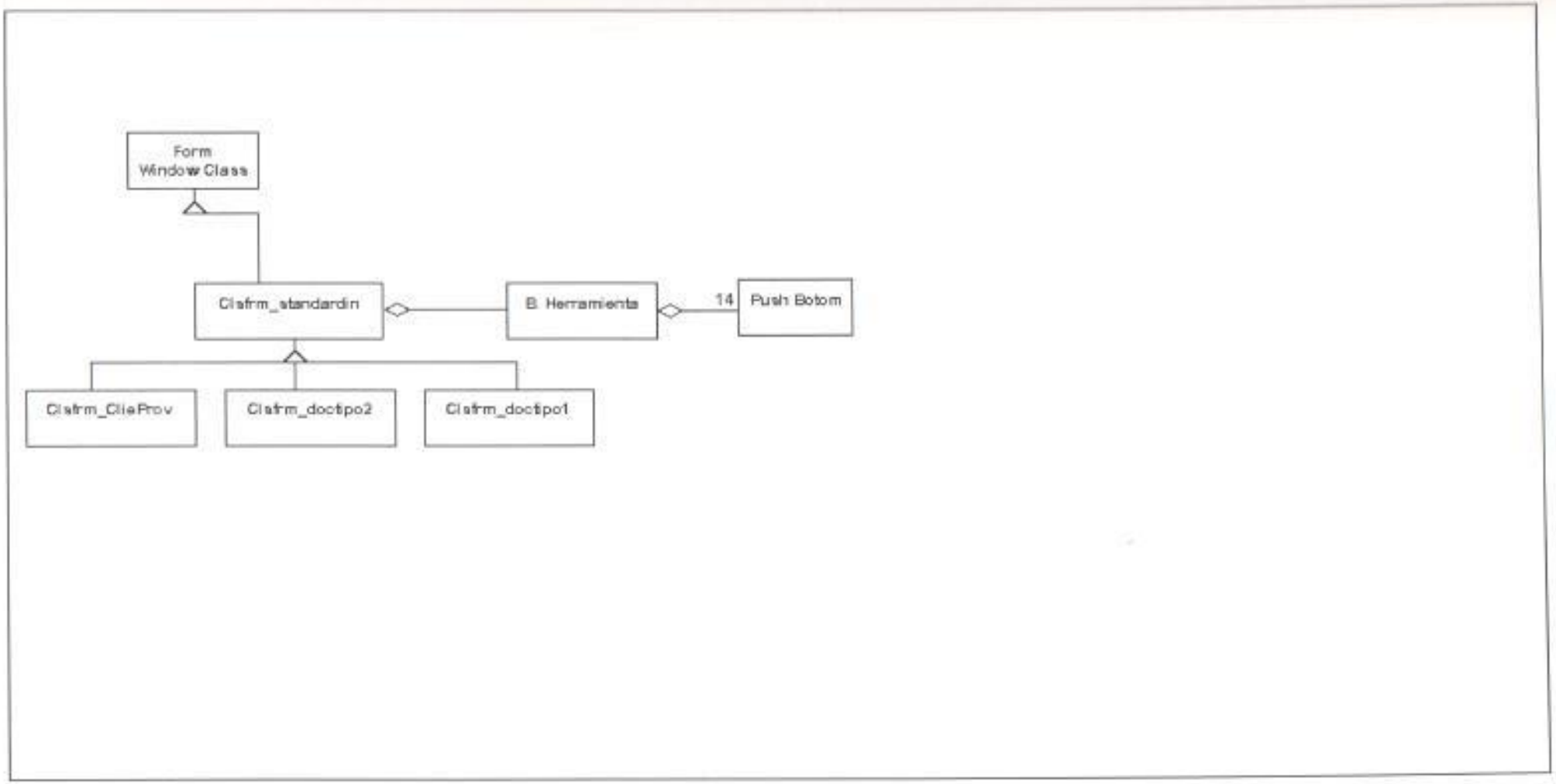
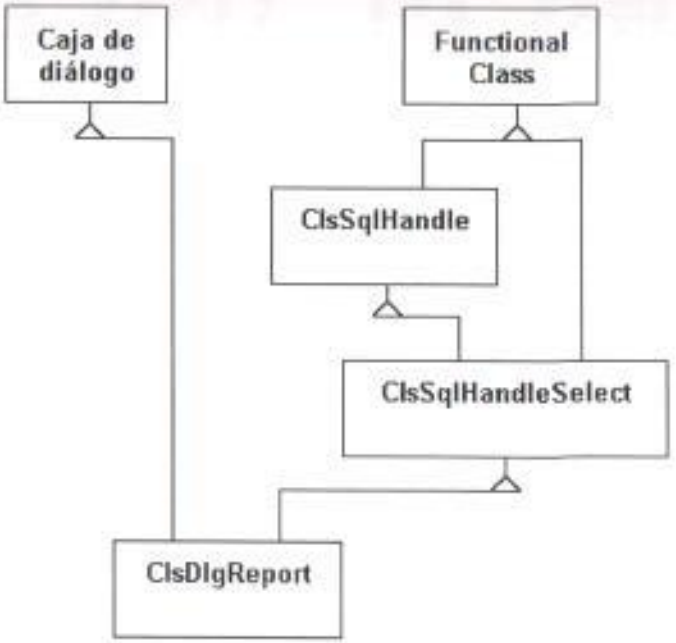


Table
Window Class
Clsfrm_standardout
B. Herramienta
7
Push Botom
Clstbl_ClePro
Clstbl_doctipo1
Clstbl_doctipo2


Form
Window Class
Clsfrm_standardin
B. Herramienta
14
Push Botom
Clsfrm_ClieProv
Clsfrm_doctipo2
Clsfrm_doctipo1


Caja de diálogo
Functional Class
ClsSqlHandle
ClsSqlHandleSelect
ClsDlgReport

# 8. DISEÑO DE CLASES

## ESPECIFICACIÓN DE CLASES

**Nombre:**

***ClsSqlHandle***

**Tipo:**

Functional Class

**Descripción:**

Clase base para todas las clases Sql handle. Su función miembro InitializeClass() debe ser llamada para inicializar el nombre de la base de datos, el nombre del usuario, y la clave de acceso. Se asume que la aplicación se conecta ***A UNA SOLA BASE DE DATOS*** a la vez.

**Derivada de:**

***Functional Class***

**Variables:**

**Clase**

| Nombre | Tipo | Descripción |
|---|---|---|
| strSqlDatabase | String | Nombre de la Base de datos. |
| strSqlUser | String | Nombre del usuario. |
| strSqlPassword | String | Cadena clave de ingreso. |

**Instancia**

| Nombre | Tipo | Descripción |
|---|---|---|
| sqlHandle | Sql Handle | Conexión de SQL a Base de datos. |
| strSQLStatement | String | Expresión SQL. |
| bConnected | Boolean | Registro de éxito de operación. |

**Funciones:**

***InitializeClass()***

**Descripción:**

Esta función debe ser llamada ***UNA VEZ POR APLICACIÓN.*** Inicializa las variables clase de la clase.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| strSqlDatabaseParm | String | Nombre de la Base de datos. |
| strSqlUserParm | String | Nombre del usuario. |
| strSqlPasswordParm | String | Cadena clave de ingreso. |

**Variables:**

**Locales**

**Estáticas**
**Retorna:**
**Error:**

***Initialize()***
**Descripción:**

Esta función debe ser llamada una vez por instancia para inicializar la variable de instancia strSQLStatement. Se asume que la función InitializaClass() ha sido llamada antes.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| strSQLStatement | String | Expresión SQL |

**Variables:**
**Locales**
**Estáticas**
**Retorna:**
**Error:**

***Error()***
**Descripción:**

Esta función despliega el texto de un error en una cuadro de mensaje. Una clase derivada de ***ClsSqlHandle*** puede sobreescribir esta función definiendo su propia función de error.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| strMessage | String | Texto de mensaje |

**Variables:**
**Locales**
**Estáticas**
**Retorna:**
**Error:**

***Connect()***
**Descripción:**

Esta función es usada para inicializar las variables de sistema SqlDatabase, SqlUser, SqlPassword y, llama a SqlConnect(). Retorna verdadero si la operación fue exitosa, y falso en caso contrario.

**Parámetros:**
**Variables:**
**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| strSqlDatabaseTemp | String | |
| strSqlUserTemp | String | |

| | |
|---|---|
| strSqlPasswordTemp | String |

**Estáticas**

**Retorna:**

Boolean

**Error:**

***SetIsolationLevel()***

**Descripción:**

Esta función puede ser usada para fijar el nivel de aislamiento para todos los cursores de la aplicación. ReadRepeatability es el valor por omisión para SQLWindows.

**Parámetros**

| Nombre | Tipo | Descripción |
|---|---|---|
| strIsolation | String | Registra nivel de aislamiento |

**Variables:**

**Locales**

**Estáticas**

**Retorna:**

Boolean

**Error:**

***Prepare()***

**Descripción:**

Esta función prepara la expresión SQL

**Parámetros:**

**Variables:**

**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| bPrepared | Boolean | Registro de éxito de operación. |

**Estáticas**

**Retorna:**

Boolean

**Error:**

Call ..Error('No puede preparar la expresión: ' || strSQLStatement)
Retorna FALSO

***Execute()***

**Descripción:**

Esta función es llamada para ejecutar la expresión SQL.

**Parámetros:**

**Variables:**

**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| bExecuted | Boolean | Registro de éxito de operación. |

**Estáticas**

**Retorna:**

Boolean

**Error:**

Call ..Error('No puede ejecutar la expresión: ' || strSQLStatement)
Retorna FALSO

***Commit()***

**Descripción:**

Esta función es llamada para fijar los cambios en la base de datos.

**Parámetros:**

**Variables:**

**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| bCommited | Boolean | Resultado de operación. |

**Estáticas**

**Retorna:**

Boolean

**Error:**

Call ..Error('No puede fijar los datos de la expresión: ' || strSQLStatement)
Retorna FALSO

***Disconnect()***

**Descripción:**

Esta función es llamada para desconectar el Sql Handle de la base de datos, si es que esta conectado.

**Parámetros:**

**Variables:**

**Locales**

**Estáticas**

**Retorna:**

Boolean

**Error:**

Call ..Error('No puede desconectar: ' || strSqlDatabase)
Retorna FALSO

**Nombre:**

***ClsSqlHandleSelect***

**Tipo:**

Functional Class

**Descripción:**

Esta clase es derivada de clsSqlHandle. Puede ser usada para definir un SqlHandle el cual va a ser usado para hacer un SELECT y browse (primer registro, último registro, siguiente registro, registro previo, y un registro específico) a través del conjunto resultado. La instancia puede definir una expresión SELECT específica llamando a la función Initialize().

**Derivada de:**

***Functional Class***
***ClsSqlHandle***

**Variables:**

**Clase**
**Instancia**

| Nombre | Tipo | Descripción |
|---|---|---|
| nResultSetCount | Numérico | Número de filas en conjunto resultado. |

**Funciones:**

***Execute()***

**Descripción:**

Esta función es llamada para ejecutar la expresión SELECT .Esta función sobre escribe a la función definida en la clase base clsSqlHandle.

**Parámetros:**
**Variables:**
**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| bSuccess | Boolean | Resultado de operación. |

**Estáticas**
**Retorna:**

Boolean

**Error:**

Call ..Error('No se puede obtener la cuenta del conjunto resultado para:
strSQLStatement)
nResultSetCount = 0
Retorna FALSO

***MessageBoxIfFetchError()***

**Descripción:**

Esta función despliega una caja de mensajes para las opciones del FETCH.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| nInd | Numérico | Indice de registro en conjunto Resultado de operación. |

**Variables:**

**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| strMessage | String | Resultado de operación. |

**Estáticas**

**Retorna:**

Boolean

**Error:**

***First()/Last()***

**Descripción:**

Esta función es usada para ir al primer/último registro en el conjunto resultado.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| nInd | Número Recibido | Código de retorno de la operación de captura. |

**Variables:**

**Locales**

**Estáticas**

**Retorna:**

Boolean

**Error:**

***Previous()/Next()***

**Descripción:**

Esta función es usada para ir al previo/siguiente registro en el conjunto resultado.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| nInd | Número Recibido | Código de retorno de la operación de captura |

**Variables:**
**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| bSuccess | Boolean | Resultado de operación. |

**Estáticas**
**Retorna:**

Boolean

**Error:**

Call ..Error('No puede capturar el Registro siguiente/previo')
Retorna FALSO

***FetchRow()***
**Descripción:**

Esta función es usada para capturar un registro específico dentro del conjunto resultado.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| nRowParm | Numérico | Indice de registro en conjunto Resultado de operación. |
| nInd | Número Recibido | Código de retorno de la operación de captura |

**Variables:**
**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| bSuccess | Boolean | Resultado de operación. |

**Estáticas**
**Retorna:**

Boolean

**Error:**

Call ..Error('No se puede capturar el registro número: ' ||
SalNumberToStrX( nRowParm, 0 ))
Retorna FALSO

**Nombre**:

***ClsDlgReport***

**Tipo:**

Dialog Box Class

**Descripción:**

Esta clase llama a SalReportView o a SalReportPrint y maneja los mensajes tales como SAM_ReportStart, SAM_ReportFetchNext.

**Derivada de:**

***Dialog Box Class***
***ClsSqlHandleSelect***

**Variables:**

**Clase**
**Instancia**

| Nombre | Tipo | Descripción |
|---|---|---|
| strReportTemplate | String | Nombre de archivo de reporte |
| strVariableList | String | Lista de variable de entradas al reporte. |
| strInputList | String | Lista de campos de entrada al reporte |
| nInd | Número | Código de retorno de la operación de captura de registro. |
| bOk | Boolean | Registro de operación. |

**Funciones:**

*View()*

**Descripción:**

Esta función llama a la función SAL SalReportView, que presenta preliminarmente al reporte.

**Parámetros:**
**Variables:**
**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| hWndReport | Window Handle | Conexión de Reporte a Forma. |
| nError | Numérico | Número de error. |

**Estáticas**
**Retorna:**
**Error:**

*Print()*

**Descripción:** Esta función llama a la función SAL SalReportPrint, que presenta a impresora el reporte.

**Parámetros:**

**Variables:**

**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| hWndReport | Window Handle | Conexión de Reporte a Forma. |
| nError | Numérico | Número de error. |

**Estáticas**

**Retorna:**

**Error:**

*Initialize()*

**Descripción:** Esta función inicializa strReportTemplate, strVariableList, strInputList, y llama a las funciones Initialize y Connect de clsSqlHandleSelect.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| strSQLParm | String | Expresión SQL |
| strReportTemplateParm | String | Nombre de archivo de reporte |
| strVariableListParm | String | Lista de variable de entradas |
| strInputListParm | String | Lista de campos de entrada. |

**Variables:**

**Locales**

**Estáticas**

**Retorna:**

**Error:**

**Nombre**:
***ClsGenResetDirty (Clase Abstracta)***

**Tipo:**
General Window Class

**Descripción:**
Esta general window class se blanquea a sí mismo después de recibir el mensaje PM_Initialize. Además, sobre SAM_Validate y SAM_AnyEdit fija bFormDirty a TRUE colocando el mensaje PM_Dirty en la cola de mensajes de hWndForm.

**Derivada de:**
***General Window Class***

**Variables:**
**Clase**
**Instancia**
**Funciones:**

**Nombre**:
***ClsDfResetDirty***

**Tipo:**
Data Field Class

**Descripción:**
Esta clase data field se inicializa a sí mismo sobre la recepción del mensaje PM_Initialze. También, sobre SAM_Validate, fija bFormDirty a TRUE.

**Derivada de:**
***Data Field Class***
***ClsGenResetDirty***

**Variables:**
**Clase**
**Instancia**
**Funciones:**

**Nombre**:

***ClsSqlHandleSGC***

**Tipo:**

Functional Class

**Descripción:**

Esta clase es derivada de clsSqlHandle definida en (SQLHANDL.APL). Sobreescribe Error.

**Derivada de:**

***Functional Class***
***ClsSqlHandle***

**Variables:**

**Clase**
**Instancia**

**Funciones:**

*Error()*

**Descripción:**

Esta función sobre escribe la función Error de clsSqlHandle. Esta función obtiene el número de error del último error, el texto de error, la causa y el remedio.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| strMessage | String | Texto de error. |

**Variables:**
**Locales**
**Estáticas**
**Retorna:**
**Error:**

**Nombre**:

***ClsSqlHandleSelectSGC***

**Tipo:**

Functional Class

**Descripción:**

Esta clase es derivada de clsSqlHandleSelect definida en (SQLHANDL.APL). Define unas nuevas funciones miembro SetParameter, y GetResultSetCount, y sobreescribe Error.

**Derivada de:**

***Functional Class***
***ClsSqlHandleSelect***

**Variables:**

**Clase**
**Instancia**

**Funciones:**

***SetParameter()***

**Descripción:**

Esta función es usada para fijar un parámetro de la Base de datos tal como DBP_FETCHTHROUGH o DBP_PRESERVE.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| nParameterParm | Numérico | Parámetro de la Base de datos a ser fijado. |
| nSetting | Numérico | Valor numérico del parámetro. |
| strSetting | String | Valor string del parámetro. |

**Variables:**
**Locales**
**Estáticas**
**Retorna:**
**Error:**

***GetResultSetCount()***

**Descripción:**

Esta función simplemente retorna la variable de instancia ResultSetCount.

**Parámetros:**

**Variables:**
**Locales**
**Estáticas**
**Retorna:**

Numérico

**Error:**
***Error()***
**Descripción:**

Esta función sobreescribe la función Error de clsSqlHandle. Esta función despliega el número de error del último error, el texto del error , su causa y su remedio, llamando a la función HandleSQLError que extrae los argumentos del Error SQL desde wParam y lParam.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| strMessage | String | Texto de error. |

**Variables:**
**Locales**
**Estáticas**
**Retorna:**
**Error:**

## Nombre:

***Clsfrm_standardin***

## Tipo:

Form Window Class

## Descripción:

Esta clase es construida para definir los controles comunes de una forma de entrada o consulta de datos. Define específicamente una barra de herramientas con 14 Push Botom, los que incluyen botones de movimiento dentro del conjunto resultado, botones para gestión de registros dentro del conjunto (Nuevo, Inserción, Actualización, y Eliminación), push botom para imprimir datos comunes de la forma (instancia) correspondiente, y un push botom para abandonar la ventana. Todos con el envío de mensajes a la clase base.

## Derivada de:

***Form Window Class***

## Variables:

**Clase**

| Nombre | Tipo | Descripción |
|---|---|---|
| hSqlFormSelect | clsSqlHandleSelectSGC | Conexión para expresión SELECT. |

| | | |
|---|---|---|
| hSqlFormSearch | clsSqlHandleSelectSGC | Conexión para búsqueda por código. |
| hSqlFormUpdate | clsSqlHandleSGC | Conexión para expresión UPDATE. |
| hSqlFormInsert | clsSqlHandleSGC | Conexión para expresión INSERT. |
| hSqlFormDelete | clsSqlHandleSGC | Conexión para expresión DELETE. |

**Instancia**

| Nombre | Tipo | Descripción |
|---|---|---|
| bFormDirty | Boolean | Cambios sobre campos de la forma. |
| nRowNumber | Numérico | Número de registro dentro del conjunto resultado. |
| bInsert | Boolean | Si es verdadero se va ha insertar un nuevo registro. |
| strSelectForm | String | Expresión SQL para SELECT. |
| strSelectSearch | String | Expresión SQL para búsqueda por código. |
| strUpdateForm | String | Expresión SQL para UPDATE. |
| strInsertForm | String | Expresión SQL para INSERT. |
| strDeleteForm | String | Expresión SQL para DELETE |
| bOk | Boolean | Resultado de operación sobre captura de registro (First, Last,..). |

## Funciones:

***OkToLoseChangesIfAny()***

**Descripción:**

Mira si la forma esta con datos (puede haber cambios sobre los data fields). Para lo cual averigua el valor de la variable de instancia bFormDirty. Si los hay pregunta al usuario si quiere perder los cambios.

**Parámetros:**

**Variables:**

**Locales**

**Estáticas**

**Retorna:**

Boolean

**Error:**

***BeginOperation()***

**Descripción:**

Esta función pone el cursor en espera y fija el texto de la barra de status de la MDI.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|

| | | |
|---|---|---|
| strText | String | Texto para la barra de status. |

**Variables:**
**Locales**
**Estáticas**
**Retorna:**
**Error:**
***EndOperation()***
**Descripción:**

Esta función restaura al cursor en espera a su estado normal y limpia la barra de status de la MDI.

**Parámetros:**
**Variables:**
**Locales**
**Estáticas**
**Retorna:**
**Error:**

***DisableClose()***
**Descripción:**

Esta función desabilitará el ítem Close desde el menú system de hWndParm.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| hWndParm | Window Handle | Handle a la ventana. |

**Variables:**
**Locales**

| Nombre | Tipo | Descripción |
|---|---|---|
| hSystemMenu | Numérico | Número de ítem del menú de la ventana. |

**Estáticas**
**Retorna:**
**Error**

**Nombre**:

***Clstbl_standardout***

**Tipo:**

Table Window Class

**Descripción:**

Esta clase es construida para definir los controles comunes de una tabla de consulta de datos (ventana para BROWSE). Define específicamente una barra de herramientas con 7 Push Botom, los que incluyen botones de movimiento dentro del conjunto resultado, push botom para imprimir datos comunes de la forma (instancia) correspondiente, un push botom para la búsqueda de un registro por código, y un push botom para abandonar la ventana. Todos con el envío de mensajes a la clase base.

**Derivada de:**

***Table Window Class***

**Variables:**

**Clase**

| Nombre | Tipo | Descripción |
|---|---|---|
| hSqlFormSelect | clsSqlHandleSelectSGC | Conexión para operación SELECT. |
| hSqlFormSearch | clsSqlHandleSelectSGC | |
| hSqlFormUpdate | clsSqlHandleSGC | Conexión para expresión UPDATE. |
| hSqlFormInsert | clsSqlHandleSGC | Conexión para expresión INSERT. |
| hSqlFormDelete | clsSqlHandleSGC | Conexión para expresión DELETE. |

**Instancia**

| Nombre | Tipo | Descripción |
|---|---|---|
| strSelectAllTable | String | |
| bPopulateTbl | Boolean | |
| hWndColumn | Window Handle | |
| nFocusRow | Numérico | |
| bBusqueda | Boolean | |
| Dato_Busqueda | String | |
| indice_busqueda | Numérico | |
| indice_anterior | Numérico | |
| strSelectSearch | String | |
| bOk | Boolean | |
| Indice_Registro | Numérico | |

## Funciones:

***BeginOperation()***

**Descripción:**

Esta función pone el cursor en espera y fija el texto de la barra de status de la MDI.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| strText | String | Texto para la barra de status. |

**Variables:**
**Locales**
**Estáticas**
**Retorna:**
**Error:**

***EndOperation()***

**Descripción:**

Esta función restaura al cursor en espera a su estado normal y limpia la barra de status de la MDI.

**Parámetros:**
**Variables:**
**Locales**
**Estáticas**
**Retorna:**
**Error:**

***DisableClose()***

**Descripción:**

Esta función desabilitará el ítem Close desde el menú system de hWndParm.

**Parámetros:**

| Nombre | Tipo | Descripción |
|---|---|---|
| hWndParm | Window Handle | Handle a la ventana. |

**Variables:**
**Locales**

| **Nombre** | **Tipo** | **Descripción** |
|---|---|---|
| hSystemMenu | Numérico | Número de ítem del menú de la ventana. |

**Estáticas**
**Retorna:**
**Error**

# 9. CONCEPTOS BASICOS

## 9.1 PILARES DEL DESARROLLO DE APLICACIONES ORIENTADAS A OBJETOS

El desarrollo de aplicaciones con el enfoque orientado a objetos introduce nuevos conceptos, que cambian la forma en la que se tiene que estudiar una situación especifica. Así, este nuevo paradigma se desenvuelve bajo los siguientes elementos:

**Clases**
**Encapsulamieto**
**Herencia**
**Polimorfismo**
**Comúnicación por mensajes**

### 9.1.1 CLASES

Una clase es un molde, el cual define las propiedades o caracteristicas, relaciones y el comportamiento de un objeto, pero no es el objeto en si. En otras palabras, una clase descibe un grupo de objetos. Cuando se crea una instancia de la clase es cuando se tiene un objeto.

### 9.1.2 CLASES SQLWINDOWS

En el caso de SQLWinodws, las clases son moldes de objetos utilizados en SQLWindows, como Form Windows, Dialog Box, Data Field, Combo Box, etc.

Las clases nos permiten generar código 100% reutilizable, ya que el código para resolver un problema se coloca en la clase y automáticamente todas las instancias de esa clase también resuelven el problema, por ejemplo considere que necesita capturar dos fechas la de ingreso y la de pago, y requiere que estas fechas sean de días habiles, descartando sabados, domingos y dias de asueto; en lugar de validar en cada Data Field que cumpla con tal condición , mejor se crea una clase denominada clsdfDiaHabiel y en ella se coloca el código de la validación , al crear dfFechaIngreso y dfFechaPago como instancias de clsdfDiaHabil automáticamente contaran con el código de la validación

### 9.1.3 ENCAPSULAMIENTO

Existen varios principios para comprender y utilizar correctamente los objetos, el primero de ello denominado ENCAPSULAMIENTO de datos consiste en que un objeto puede modificar alguno de sus atributos, pero no podrá manipular los atributos de otro objeto.

Regresando al ejemplo de la cadena de restaurantes, cada sucursal tiene su menú, y solo puede modificar el menú de esa sucursal.
Aunado al concepto de encapsulamiento aparece otro: Comúnicación por mensajes, el mecanismo por el cual un objeto solicita una acción a otro objeto, en nuestro ejemplo la dirección general de la cadena de restaurantes seria otro objeto, que en determinado momento decide que los precios del menú deben incrementarse pero la dirección general
no modificara directamente los precios de los menús de todas las sucursales, sino que enviara a cada sucursal una notificación para que realicen el aumento respectivo, cada sucursal determinara como reimprimira sus cartas con los nuevos precios.

### 9.1.4 HERENCIA

Una propiedad muy importante de la programación orientada a objetos consiste en poder combinar propiedades de dos clases o más en una clase, a lo que se denomina HERENCIA.

La Herencia se basa en generar una nueva clase a partir de una o varias clases ya existentes, la nueva clase posee automáticamente las mismas propiedades y comportamiento de las clases que la generaron.

Suponga que tiene dos clases: la primera DiaHabil valida que el dato sea una fecha habil, y la segunda: FontColor permite definir el tipo de letra y color que tendrá un dato al ser desplegado en pantalla; seria deseable contar con ambas cosas en un solo objeto, es decir
un objeto que valide el dato y sobre el cual haya un control del tipo de letra y color para desplegarlo, para eso lo único que se requiere es crear una nueva clase FechaHabil y definir que sus antecesores son las clases DiaHabil y FontColor.

Suponga que en un futuro el algoritmo para validar que una fecha sea habil cambia, si eso sucede solo necesita modificar el código de la clase DiaHabil, la clase FechaHabil automáticamente registrara el cambio gracias al mecanismo de la herencia.

### 9.1.5 POLIMORFISMO

La habilidad de que diferentes objetos respondan al mismo evento en diferente forma. Esta propiedad se utiliza con los objetos estandar de SQLWindows, tome por ejemplo el caso de los pushbottons, todos los pushbuttons reciben el mensaje SAM_Click y sin embargo un boton desplegara el siguiente registro, otro pushbutton mostrara el registro anterior, y otro más desplegara el ultimo registro, aunque reciben el mismo mensaje realizan diferentes acciones.

El polimorfismo fácilita la independencia entre objetos, ya que cuando un objeto va a enviar un mensaje a otro objeto no necesita conocer la naturaleza del receptor, simplemente le envia el mensaje y el objeto responderá en la forma apropiada.

## 9.2 BENEFICIOS DE OOP

***- FÁCILIDAD DE USO***
Simplifican el desarrollo de aplicaciones al crear entidades que tratan con un problema especifico

***- REUSABILIDAD***
Las clases creadas y utilizadas en una aplicación pueden volverse a utilizar en otra aplicación

***- FÁCIL MANTENIMIENTO***
Al aislar las funciones en clases especificas un problema se puede identificar fácilmente y al realizar la modificación en la clase automáticamente las instancias de esa clase (objetos)
se corrigen también.

***- REDUCCIÓN DE COSTOS DE DESARROLLO***
Como consecuencia directa de los puntos anteriores, y con ello incrementa la productividad.

## 9.3 DISEÑO DE APLICACIONES CON OO

Cuando se utiliza OOP para el desarrollo de aplicaciones se debe modificar la forma en que se resuelven problemas desde el diseño de la aplicación :

1. Como primer paso se definene las clases generales de la aplicación basandose en la función de las clases, cada clase se debe diseñar para realizar una actividad especifica.

2. Con la lista de clases se determina como deben combinarse (derivarse) estas clases para generar la funcionalidad de clases más completas.

# 10. DEFINICIÓN DE CLASES

## 10.1. TIPOS DE CLASES

**Window Class**
Estan basadas en un objeto gráfico de SQLWindows, como DataField, Form Window, Pusbbutton, etc. posee todos los atributos del objeto gráfico (como tamaño, color, tipo de letra, etc) tiene la facultad de recibir mensajes (SAM_, PAM_, VTM_).

**Functional Class**
No estan basadas en ningun objeto gráfico de SQLWindows, solo puede definir funciones y variables. No reciben mensajes.

**Class Definitions**
Las clases se definen en la sección Class Definitions de Global Declarations, y se pueden crear de tres formas distintas:
- Seleccionando el tipo de clase desde el Outline Options
  Windows (Asistente).
- Utilizando el editor de clases, o
- Tecleando directamente en el outline.

El editor de clases es utilizado cuando se deseen crear nuevas Windows Class que tenga o no herencia.
No se pueden crear Nuevas functioanl class, solo se pueden crear copiando la functional class ya existentes.

## 10.2 ELEMENTOS DE UNA CLASE

**Descripción :** Texto documentativo de la clase Derived From: Contiene la lista de las clases de las que deriva esta clase.
**Instance Variables:** Define las variables de las instancias de la clase, estas variables perteneceran exclusivamente al objeto, es decir el valor de las mismas es único para cada instancia.
**Class Variables:** Define las variables de la clase, estas variables estaran accesibles a todas las instancias de lamisma clase, es decir el valor de las variables de la clase es

el mismo para todas las instancias de la misma clase, es decir, el valor de las variables
de clase es el mismo para todas las instancias de esa clase.
**Functions:** Define las funciones de la clase estas funciones se definen como las funciones internas pero en lugar de ser funcines globales son funcines de la clase.

## 10.3 CLASS VARIABLES VS INSTANCE VARIABLES

Las instancias de la clase comparten la class variable, de tal forma que al final del proceso, si las instancias preguntan por la variable de clase esta tendrá el valor comun. Por otro lado
las variables de instancia no son compartidas cada una maneja un valor independiente.

# 11. WINDOW CLASES

## 11.1 TOP-LEVEL WINDOWS CLASS

MDI Window
Form Window
Dialog Box
Top Level table Window
Quest Window

**- ACTIVE CHILD WINDOWS CLASS (RECIBEN MENSAJES)**
Data Field
Pushbutton
Option Button
Combo Box
Picture
Column
Radio Button
Multiline
Child Table
Scroll Bar (Horizontal y Vertical)
Custom Control
Check box
List Box
Quest Child Windows

**- PASSIVE CHILD WINDOWS CLASS (NO RECIBEN MENSAJES)**
Background
Group Box
Frame
Line

Las Window Class se clasifican en dos grandes grupos:
- Top Level Window Class
- Child Window Class

Los Child Window Class por otra parte se subdividen en:
- Active Child Window Class: Poseen una sección Message Actions
- Passive Child Window Class: NO tiene una sección de Message Actions

Al igual que los objetos las clases cuentan con el customizer para definir atributos visuales de las instancias de la clase.
Así por ejemplo si se define que la clase clsDfCampo tendrá el font Arial al tamano de 8 puntos, todas las instancias de esta clase tendrán ese font y tamaño.
Sin embargo no es obligatorio para la instancia utilizar las características de la clase de instancia puede redefinir estas características, a este proceso se denomina Override concepto de override aplica tanto para comportamiento como para atributos, el concepto de override se vera con más detalle posteriormente.

## 11.2 CREANDO LOS OBJETOS

Las instancias de Window Classes se denominan User-defined Windows Objects (Objects Windows definidos por el usuario), para crearlos se sigue el mismo procedimiento que para crear los objetos estandar de SQLWindows.
Desde el Outline Optiones Windows, seleccione la clase de la instancia.
Desde el Tool Pallete seleciones el tipo de objeto, en la parte superior aparecerá una lista de las clases disponibles, elija la apropiada y trace la localización del objeto en la ventana. En el caso de una Top-Level Window aparecerá un menú extra para escoger la clase.

## 11.3 UBICACIÓN EN EL OUTLINE

Los objetos se ubican exactamente igual que los objetos estandar de SQLWindows como se puede apreciar se asemejan a objetos estandar pues también tienen su customizer y sección de Message Actions.

## 11.4 OVERRIDE DE MENSAJES

Es muy importante observar que tanto la clase como la instancia de la clase tienen una sección para atender mensajes, por lo que tanto la clase como el objeto podrian atender el mismo mensaje y en ese caso la instancia estaria haciendo override sobre la atención del mensaje.

# 12. FUNCTIONAL CLASSES

## 12.1 USO

- Para aislar un comportamiento o caracteristica
Windows Class
- Para declarar variables definidas por el usuario

A diferencia de las Windows Class, las Funcional Class no estan basadas en gráfico en objetos gráficos su próposito muchas veces es el de servir a otras clases.

Existen casos en que es necesario definir varias clases funcionales, realmente no se crean sino que se combinan atributos y comportamientos de otras clases, con lo cual la instancia s hereda automáticamente los atributos y comportamientos de las clases.

Las funciones pueden utilizarse para generar otras clases funcionales Windows Class en contraparte las Windows Classes solo pueden generar Classes del mismo tipo (una clase Pushbutton puede generar otra clase no puede generar un Data Field).

## 12.2 VARIABLES DEFINIDAS POR EL USUARIO

Las variables definidas por el usuario son instancias de Funtional Classes, estas pueden ser definidas al igual que las variables normales (la única excepción es en la sección Local Variables de las funciones.

En forma analoga a las Windows Classes las Functional Classes aparecerán en la Lista de Tipos de Datos, es decir aparecerán con los tipos estandar:
Boolean, Date/Time, etc.

Las variables definidas por el usuario pueden utilizarse para definir tipos de datos más sofisticados, o bien cuado no haya una liga entre un objeto gráfico y las funciones o variable de una Functional Class.

Realmente la única razón para declarar una variable de usuario es cuando no existe un Window Class derivada de la funtional class y sin embargo se requiere el acceso a variables o funciones de la functional class.

## 12.3 FUNCTIONAL CLASS SIN FUNCIONES

Las clases funcionales pueden ser tan sencillas como solamente almacenan las variables para administrar cierta labor. Se podrá pensar dado que seria mejor crear una clase que contenga ambas caracteristicas en algunos casos es valido pero no es recomendable, tener las funcionalidades separadas proporciona modularidad, fácilmente se puede derivar una clase de las dos o solamente de una.

## 12.4 FUNCTIONAL CLASS CON FUNCIONES

### *FUNCIONES DE LA CLASE*

El cuerpo de la función de una clase sigue los mismos lineamientos que las funciones globales, las funciones de la clase pueden accesar en forma directa las variable de instancia y de clase, ya que se encuentran en el mismo contexto..

Por otra parte las funciones al ser definidas para la clase no pueden tratarse como variables globales o generales, sino solo dentro de una instancia de la clase como se vera más adelante.

### *LLAMADO DE FUNCIONES*

```
Definición   Functional Class:fcDBSelect
de           Functions
Clase            Function: Connect

         Form Windows Class:clsFrmForma
           Derived From: fcDBSelect

Instancia    clsFrmForma: frmAutores
de           Message Actions
la               On SAM_Create
Clase                Set .....
                   Set .....
                   Set .....
```

Al derivar una clase de otra, la nueva clase hereda las funciones de la clase; por ejemplo frmAutores es una instancia de la clase clsFrmForma por la herencia puede accesar las variables de la clase fcDBSelect y puede invocar a la función Conecta( ).

# 13. JERARQUÍA DE CLASES

## 13.1 HERENCIA SENCILLA

La herencia permite que las propiedades y comportamiento de una clase pasen a ser parte de otra clase, la clase que proporciona sus propiedades y comportamientos se denomina Clase Base, y la que las hereda se llama Clase Derivada.

Cuando se crea una clase derivada esta poseera las estructuras de datos, funciones y código de sus clases base, y además sus propiedades y comportamientos, es decir la clase derivada es una versión aumentada de la clase base, de esta forma se pueden crear clases más especializadas.
La herencia sencilla es la más básica de las derivaciones, en ella una clase se deriva de otra, y automáticamente posee atributos y funciones como si estuvieran definidas en ella misma. La nueva clase agrega datos y comportamiento generando así una clase más especializada.

## 13.2 JERARQUÍAS

Cuando se tiene varios niveles de derivación de clases se dice que se tiene una jerarquía de clases, en la cual existen dos tipos de clases bases.

Clase Base Directa: Cuando es base de la clase del siguiente nivel.
Clase Base Indirecta: Cuando existe una relación de herencia a través de ota clase.

En el caso de varios niveles de clases se siguen los mismo principios de herencia, la clase del nivel más bajo hereda las propiedades y comportamiento de todas las clases base en los niveles superiores.

Si una clase en el nivel más bajo invoca a una función , lo primero que se hara es buscar la definición de la función en la definición de la clase, si no la encuentra buscara en la definición de la clase base directa, y si no la encuentra buscara en el siguiente nivel y asi sucesivamente hasta localizar o detectar que no existe en ninguna de las clases de la jerarquía.

## 13.3 HERENCIA MÚLTIPLE

Por otro lado cuando una clase se deriva de dos o más clases, se tiene herencia múltiple la clase derivada hereda los comportamientos de las dos (o más) clases base, combinandolas en una nueva clase.

La herencia múltiple permite una gran flexibilidad, en muchas ocasiones la herencia sencilla no satisface todas las necesidades de derivación

Al igual que la herencia sencilla, se trata de crear una nueva clase más especializada que posee las propiedades y comportamientos de sus clases base más sus propias propiedades y comportamientos.

## 13.4 DERIVANDO CLASES

La derivación de clases se puede realizar por dos mecanismos:
1. Utilizando el editor de clases
2. Tecleando directamente en el outline

En el caso de que una Window Class se derive de más de una clase se aplican los siguientes criterios.
a) La nueva Window Class hereda los atributos de la primera clase base definida en la lista
b) Se pueden adecuar los atributos de la clase derivada, pasando por alto los atributos heredados de las clase base.
Esto es, suponga que tiene dos Form Window Class (clsFrmA y clsFrmB), en uno ha definido un menú de tres opciones opciones y en otra un menú de cinco, si genera una nueva Form Windows Class cuyas clases bases sean las anteriores. la nueva clase heredara los atributos de la primera clase enlistada, si primero se enlista clsFrmA, la nueva clase tendrá un menú de tres opciones, si por el contrario primero se lista la clase clsFrmB, tendrá un menú de cinco opciones.
El mismo criterio se aplica para color, tamaño, titulo, font, etc.

## 13.5 CALIFICACIÓN DE REFERENCIAS

Para hacer referencia a ua variable o función de una clase se debe calificar la referencia existen cuatro formas de hacer referencia, algunas son muy sencillas y directas otras se necesitan cuando se desea invocar una función de un objeto diferente al objeto actual.

**Sin calificación**
Se pueden utilizar tanto en clases como en instancias, siempre que no existe confusión en cuanto a la ubicación de la variable o función , es decir siempre que esta sea localizada en la clase o en una clase base.

***Calificación por clase***
Se utiliza desde una clase, se utiliza para especificar a cual clase pertenece la variable o función, sobre todo en el caso de que la variable o función resida en más de una clase.

***Calificación completa de la clase***
Se puede utilizar tanto en la clase como en la instancia de la clase, se necesita cuando se hace uso del Window Handle del objeto (como HWndItem o hWndFrom) sin la calificación se genera un error.

***Calificación por objeto***
Se utiliza solo en la instancia en ella se especifica a que objeto pertenece la variable o función

***Calificación completa del objeto***
Se utiliza solo en la instancia y especifica a que ventana Top Level pertenece el objeto, se utiliza para accesar funciones de ventana diferente a la actual o en funciones globales.

## 13.6 REGLAS DE DERIVACION

| | |
|---|---|
| Functional Class | Functional Class |
| Functional Class | Window Class |
| Window Class* | General Window Class |
| Window Class | Window Class |

* del mismo tipo

SQLWindows soporta tanto herencia sencilla como herencia múltiple y se pueden crear jerarquías de clases muy diversas, deben seguirse las siguentes reglas:
A. Una clase functional puede ser clase base de otra clase functional de una Window Class o de una General Window Class
B. Una Window Class puede ser clase base de otra Window Class siempre y cuando sean del mismo tipo. (Data Field-Data Field, Frame-Frame, etc.)
C. Una General Window Class puede ser clase base de una Window Class sin importar su tipo.

## 13.7 GENERAL WINDOW CLASS

Las General Windows Class son clases que reciben mensajes (como las Window Class) pero no estan basadas en un objeto gráfico de SQLWindows.

Su principal próposito es servir como base para otras clases (como las clases funcionales) ya que no pueden crear instancias de General Window Clases, no se puede accesar al customizer ni poseen sección de Tool Bar, Menú o Contens.

Suponga que dos window classes de diferente tipo deben atender el mismo evento de la misma forma (SAM_Cretae por ejemplo), en lugar de colocar el código en las dos clases puede definir una General Window Class que atiende el mensaje convierliendose en clase base de las clases originales.

Descripción : Texto documentativo de la clase
Derived From: Lista de las clases bases de esta clase, solo puede listar clases funcionales o General Windows classes
Class Variables: Variables de la clase
Instance Variables: Variables de las instancias

Functions: Funciones de la clase
Message Actions: Atención de mensajes de la clase.

Una General Winodw Class se utiliza cuando dos o más clases de diferente tipo atenderán mensajes en la misma forma. El objeto que use la GWC debe debe tener en la sección del Derived From, la GWC usada. La implementación requiere la GWC por la atención de mensajes, no es posible resolver el problema con una Functional Class.

# 14. POLIMORFISMO

## 14.1 BINDING

La resolución de una llamada a un código (función o atención de mensaje)

**BINDING ESTATICO**
(static-binding, early-binding, compile-time-binding)

Cuando la resolución de la llamada se realiza en tiempo de ejecución

**BINDING DINAMICO** (dynamic-binding, late-binding, run-time-binding)

Cuando la resolución de la llamada se realiza en tiempo de ejecución
Cuando una herramienta de desarrollo copila una aplicación debe determinar donde se encuentran las funciones o el código que va a ejecutar, a este próposito de le denomina binding.

Las herramientas tradicionales solo podian resolver el llamado en tiempo de compilación a lo cual se le denomina Binding Estatico, ya que el código o la función siempre se localizaban en el mismo lugar.

Con el advenimiento de la tecnologia OOP se hizo necesario poder realizar un Binding en tiempo de ejecución , ya que ahora se podia tener una clase general que debia hacer algun trabajo como podría ser imprimir, pero si el objeto es un editor de texto caracteres, mientras que si es un programa gráfico debera trazar líneas: sin embargo al objeto solo se le envia la petición (imprimir) y el objeto decidirá como realizar la labor.

En las herramientas tradicionales se debia definir una función o código especifico, por ejemplo si el programa debia trazar figuras geometricamente se requeria definir una función TazaTriangulo() para dibujar triangulos, una función TrazaCuadrado() para dibujar los cuadrados, etc.

En la programación orientada a objeto se pude tener una sola función que se llame Traza() y el objeto dependiento de su naturaleza decidira si debe dibujar un triangulo, cuadrado o cualquier otro objeto.

A la propiedad de que diferentes objetos respondan a la misma petición en forma diferente se denomina Polimorfismo, y puede ser implementado con el Binding Dinamico.

# 15. LA PROGRAMACIÓN ESTRUCTURADA VS LA POO

La programación orientada a objeto (POO) OOP (OBJECT-ORIENTED PROGRAMMING)
fue desarrollada, esencialmente, por las limitaciones que otras técnicas de programación tenian y tienen, como es el caso de la programación estructurada.
Recordemos las caracteristicas y las limitaciones más importantes de la programación estructurada.

La idea fundamental de la programación estructurada es romper un programa en unidades más pequeñas, denominadas procedimientos, funciones, subprogramas o subrutinas). Cada una de las funciones (al menos idealmente) tiene un próposito claramente definido, asi como una interfaz a las otras funciones utilizando parámetros, y las funciones pueden tener datos a los cuales no se puede acceder fuera del ambito del procedimiento.

La programación estructurada usa el método descendente y el refinamiento sucesivo, y cada una de las funciones se invocan sucesivamente. Los programas estructurados son más fáciles de escribir y mantener que los programas no estructurados.

## 15.1 OOP(OBJECT-ORIENTED PROGRAMMING)

Sin embargo, a medida que el tamaño de los programas aumenta y se hacen muy grandes y más complejos, el enfasis se pone en los tipos de datos que se procesan.
Las estructuras de datos en un programa se hacen tan imporantes como las operaciones realizadas por ellos. Los tipos de datos se procesan en muchas funciones, dentro de un programa estructurado, y cuando se producen cambios en esos tipos de datos, las modificaciones se deben hacer en cada posición que actua sobre esos tipos de datos dentro del programa. Esta tarea a parte de consumir gran tiempo, puede ser frustrante en programas que contengan millares de líneas de código y centenares de funciones. Esta circunstancia se produce esencialmente porque muchas funciones comparten datos (variables gloables). Asi por ejemplo, si se añaden nuevos datos, se necesitaran modificar todas las funciones que acceden a los datos, de modo que ellas también puedan acceder.
Se necesita, por consiguiente, un método para restingir el acceso a datos, ocultarlos de funciones no deseadas.

Otro ejemplo ques se suele presentar es que los programas estructurados, son con frecuencia, dificiles de diseñar, ya que funciones estrcuturadas de datos- no modelizan muy bien el mundo real.

Existen otros problemas, como la dificultad de crear nuevos tipos de datos (pese a que la mayoria de los lenguajes, C entre ellos, soportan tipos de datos definidos por el usuario). Por ejemplo, si tiene que trabajar con números complejos, fechas, coordenadas, etc. es dificil crearlos, pero luego aun más operar con ellos (sumarlos, restarlos, etc). El ultimo inconveniente que consideraremos se produce cuando un equipo de programadores trabajan en equipo en una aplicación en un programa estructurado, a cada programador se le asigna la construcción de una tarea especifica de funciones y tipos de datos.

Dado que diferentes programadores manipulan funciones independientes que comparten datos, los cambios que un programador hace a los datos deben ser reflejados en el trabajo del resto del equipo. Los problemas y los errores de comúnicación se reflejaran en el diseño final.

La programación orientada a objetos enfatiza en los datos, al contrario que la programación estructurada que enfatiza en los algoritmos.

POO permite organizar los datos de su programa al igual que los objetos que forman el mundo real; imaginese los departamentos de una gran compañia - ventas, contabilidad personal, técnico, etc. - o los diferentes vehiculos que se construyen en una fabrica de automoviles etcetera: todos ellos son objetos.

En POO, coches (carros), árboles, departamentos, públicaciones, se conocen en general como objetos.

Un objeto contiene en una sola entidad y con un único nombre las estructuras de datos y las acciones (procedimientos y funciones) que actuan sobre esos datos. Segun el lenguaje POO, el termino objeto puede tomas diversos nombres: objeto en Turbo Pascal 5.5, 6.0 7.0 y la clase en C++.

Así pues, una clase (objeto e general) es una plantilla o modelo que define los datos y las funciones que actuan sobre esos datos (llamdos métodos). Por ejemplo, una clase puede describir las caracteristicas fundamentales de un ejecutivo de una empresa de computadoras (nombre, titulo, salario, cargo, departamento al que pertenece, etc.), mientras que un objeto representara un ejecutivo especifico (Pepe Mortimer, fisico, 1946, jefe de departamento, etc.)

Cuando un elemento dato es miembro de una clase, se llama objeto. Se puede asemejar las clases a los tipos de datos definidos por el usuario en un lenguaje de programación tradicional, ya que los tipos de datos definidos por las clases se comportan como los tipos incorporados a un lenguaje de programación Una clase encapsula (encierra) en si misma datos y métodos (funciones) que manipulan los datos de los objetos de la clase. Si una clase es como un tipo de datos, un objeto es como una variable. Con frecuencia a los objetos se les llama como instancias, casos, modelos o ejemplos (instancia). Las funciones llamadas métodos (en Smalltalk) se suelen denominar funciones miembros en C++. Las funciones miembro definidas en una clase se conoce como ENVIO DE UN MENSAJE al objeto.

El proceso de programación en un lenguaje orientado a objetos se caracteriza por:
1. Crear clases que definan la representación y el comportamiento de los objetos
2. Crear objetos a partir de la definición de la clase.
3. Realizar la comúnicación entre objetos a través del envio de mensajes. (invocación a las funciones miembros)

## 15.2 ¿QUE COSAS PUEDEN SER OBJETOS EN POO?

La respuesta a esta pregunta solo tiene los limites que imponga su imaginación :

Elementos de entornos gráficos:
Ventana, Menús, Iconos, Raton,
Teclado.....

Estructuras de datos:
arrays, pilas, listas enlaadas, árboles,
grafos.....

Datos matemáticos:
ángulos, complejos, puntos en el plano,
figuras....

Objetos fisicos:
automoviles, aviones, casas....
.....

## 15.3 PROPIEDADES FUNDAMENTALES DE POO

Las propiedades fundamentales de POO son, además de las caracteristicas ya mencionadas:
- Abstracción de datos
- Encapsulamiento y ocultación de datos
- Herencia
- Polimorfismo
- Reutilizacion

### *ABSTRACCIÓN DE DATOS*

La abstracción fue un concepto introducido en programación estructurada.
Se define como la capacidad para examinar algo sin preocuparse de sus detalles internos. En un programa estructurado es suficiente conocer la tarea especifica que hace un procedimiento dado, y no es tan importante como se realiza esa tarea. Esta propiedad se conoce como abstracción funcional.
La abstracción de datos es a los datos lo que la abstracción funcional es a las operaciones. Con abstracción funcional es a las operaciones.
Con abstracción de datos, las estructuras y los items de datos se pueden utilizar sin tener que preocuparse sobre los detalles exactos de su realización (implementación).

Por ejemplo, los números en coma flotante son una abstracción en los lenguajes de programacion; no tiene que preocuparse de la representación binaria exacta de un número en coma flotante cuando se le asigna un valor , ni se necesita conocer los pormenores de la multiplicación binaria para multiplicar valores en coma flotante.

Lo más importante es que los números de coma flotane actuen de una manera correcta y comprensible. La abstracción de datos libera al programador de preocuparse sobre los detalles no esenciales. La abstracción de datos ha existido formalmente en todos los lenguajes de programación para elementos complicados, tales como los citados números de coma flotante. Sin embargo han sido los lenguajes modernos Modula-2, Ada y los lenguajes orientados a aobjetos (puros como Smaolltalk o híbrido como C++ y Turbo Pascal 5.5/6.0/7.0) quienes han permitido definir sus propios tipos abstractos de datos.
Tipos abstractos de datos (TAD) es un tipo de dato definido por el programador que se puede manipular de un modo similar a los tipos de datos definidos o incorporados al sistema. Al igual que con los tipos definidos al lenguaje, un TAD pertenece a un conjunto de valores legales de datos (rango) y se pueden relizar sobre esos valores un número de operaciones primitivas. Los usuarios pueden crear variables con valores cuyos rangos estan en el conjunto de valores cuyos rangos estan en el conjunto de valores legales, y pueden operar sobre esos valores las operaciones definidas.
Por ejemplo un programador experimentado puede definir una pila (stack) como un tipo de abstracto de dato, y las operaciones tipicas en la pila (meter, sacar y limpiar) como las operaciones o funciones (métodos) del tipo de dato. Los paquetes en Ada,

las unidades en Turbo Pascal y los modulos en Modula-2 son tipos abstractos de datos.

Un objeto es simplemente un tipo abstracto de datos y, por ello, las nociones de programación orientada a objetos se construyen sobre la idea de dicho tipo de datos, añadiendoles otras propiedades, como puede ser la ocultación y la reutilización de código.

## *ENCAPSULAMIENTO Y OCULTACIÓN DE LA INFORMACION*

La abstracción y el encapsulamiento son conceptos complementarios: la abstracción se enfoca a la vista exterior de un objeto y el encapsulamiento -también conocido como ocultación de la información- previene a otros objetos ver su interior, donde el comportamiento de la abstracción se ha realizado (implementado). Un concepto fundamental en las clases es la oculación de los detalles de su implementación En la practica, esto significa que cada clase debe tener dos partes: una interfaz y una implementación La interfaz de una clase se dedica solo a la vista exterior y la implementación comprende la representación de la abstracción , así como los mecanismos que realizan los comportamientos deseados.
Por ejemplo, sonsideremos una biblioteca de clase gráficos que puede tener diferentes implementaciones para difirentes tipos de pantallas de video. Un programa que utiliza clases graficas se puede escribir de tal modo que sea compilable, sin modificaciones, en cualquier computadora a la que se transporta la implemtación de la biblioteca.
De este modo, las clases suelen tener dos partes: parte pública (accesible fuera de la clase). Los datos y métodos se pueden definir indistintamente en cada parte. C++ define uan tercera sección en una clase: la parte protegida. Es aquella a la que pueden acceder no solo las funciones miembro(métodos) dentro de las misma clase, sino también funciones miembros de clases derivadas o descendientes de esa clase

## *HERENCIA*

La idea de clases conduce a la idea de herencia. En nuestra vida diaria, el concepto de clases se divide un subclases. Así, las clases de animales se dividen en: mamiferos, amfibios, insectos, ect. las clases de vehiculos se dividen en coches(carros), motocicletas, tractores y autobuses, etc. El principio en el que se basan las sucesivas subclases es que cada subclase comparte características comunes con la clase que se deriva o desciende; por ejemplo, todas las clases vehiculo tienen un motor, ruedas y un volante o millar. Además de las características comunes, cada subclase tiene sus propias características particulares; por ejemplo, número de ruedas, número de asientos, diferentes portaequipajes, etc.

En POO las clases se pueden subdividir en subclases; dicho de otro modo, una clase se puede deducir o derivar de otra clase. La clase original se denomina clase base y, merced a la propiedad de la herencia una clase derivada o descendiente puede heredar las estructuras de datos y los métodos de su clase base. Además, la nueva clase puede añadir nuevos elementos de datos y métodos a los que hereda de su clase base. Cualquier clase base(incluida una descendiente) puede tener cualquier número de clases decendientes. Mediante el mecanismo de la herencia se puede construir una jerarquía de clases que adoptará la forma típica de árbol, donde la clase base se llamará clase padre y, la clase descendiente o derivada, clase hija.

Supongamos, por ejemplo, que se desea construir un conjunto de clases que describa una biblioteca de públicaciones. Consideremos dos tipos de públicaciones: periódicos y libros. Se puede crear una clase base públicación con los datos: titulo, número de página, número de catalogo, año de públicación , precio y editor; los métodos de las públicaciones pueden ser: leer, almacenar y buscar o recuperar del depósito o biblioteca.

A continuación se crean dos clases decendientes llamadas periódico y libro. Un periódico tiene una fecha de públicación (además del año) y un número de orden y una serie de articulos: son los miembros datos de periódico. El periódico utilizará métodos: la suscripción o la compra en el quiosco de prensa. El libro incluirá como datos el nombre de su autor, el número ISBN, clase de libro (novela, poesia, técnica, ficción , etc.). Los métodos de libros pueden ser comprar en libreria y regalar. En resumen, un libro y un periódico comparten las caracteristicas comunes de una públicación y, además, tienen sus propios atributos. Así, vemos que la clase base define métodos de almacenamiento y recuperación que seran distintos en cada caso: los periódicos se almacenarán con reseñas de sus articulos, fecha de públicación y temas; los libros se registran en catalogos generales o especificos de
editoriales, etc.
Existen dos tipos de herencia: simple y múltiple.
La herencia simple, tratada anteriormente, es aquella en la que una clase solo puede tener un ascendiente.
Herencia múltiple es aquella en que un objeto puede tener más de un descendiente.
La mayoria de los lenguajes soportan herencia simple. Solo algunos incorporan la propiedad de herencia múltiple: C++(2.1/3.0) y Eiffel.
Un caso tipico de herencia múltiple es el popular termino de multimedia. Supongamos dos tipos de clases: Sonido y Gráfico; si se combinan ambos, pueden dar lugar a una nueva clase multimedia comp propiedades de ambas clases. Sin embargo. es preciso advertir que en herencia múltiple se plantean dificultades diversas, sobre todo cuando se combinan tipo
de clases, cada una de las cuales define los mismos campos o métodos.

## *POLIMORFISMO*

Es la propiedad que permite a una operación tener diferente comportamiento en objetos diferentes. En otras palabras, objetos diferentes reaccionan de modo diferente al mismo mensaje. Por ejempl, supongamos un determinado número de figuras geométricas, en las que todas ellas cmprenden el mensaje dibujar. Cada objeto reacciona a este mensaje, visualizandose a si mismo en la pantalla de video. Evidentemente, un objeto rectangulo se dibujará de modo diferente que un objeto circulo.

El polimorfismo juega un papel importante en la simplificación de la sintaxis al realizar la misma operación sobre una colección de objetos. Por ejemplo, el polimosrfismo permite dibujar todas las figuras geométricas pertenecientes a una array de figuras, mediante un bucle similar a este:

Esto es posible debido a que, con independencia de la figura geometrica exacta, cada objeto comprende el mensaje dibujar y reacciona a el de un modo apropiado a esa figura.

El polimorfismo requiere herencia; simplifica la tarea del programador, generalizando la sintaxis de comúnicación que permite tratar objetos de un tipo diferente de modo similar. El polimorfismo se aplica únicamente a los métodos heredados de una clase base. El polimorfismo significa que un método puede tener un nombre que es compartido a lo largo de una jerarquía de clases. Cada clase puede tener una implementación diferente para ese método. El nombre del método es el mismo para cada objeto; sin embargo, lo que hace es diferente.

El polimorfismo depende de la ligadura, que es el proceso por el cual un método se asocia cun una función real. Cuando los métodos polimorficos se utilizan, el compilador no puede determinar que función llamar. La función especifica llamada depende de la clase del elemento a la cual se envia el mensaje; en consecuencia, la función a llamar se determina en tiempo de ejecución Esta operación se conoce como ligadura tardia o postergada (late binding) ya que ocurre cuando el programa se esta ejecutando. La ligadura temprana, anticipada o previa (early binding) ocurre para métodos no pilimorficos; el compilador conoce exactamente cual es el método que se invoca, de este modo se puede construir una llamada directa en tiempo de compilación Aunue la ligadura previa es muy eficiente, la mayoria de los lenguajes orientados a objetos utilizan ligadura postergada para todos los
métodos.

## *REUTILIZACION*

Una vez que una clase se ha escrito, creado, depurado y comprobado, se puede distribuir a otros programadores para utilizarla en sus propios programas. Es decir, se puede utilizar como un bloque basico para construir programa. Esta operación se llama reutilización Los programas orientados a objetos se construyen a partir de componentes reutilizables de software. Esta operación es similar al modo en que una

biblioteca de funciones en un lenguaje procedimental se puede incorporar en diferentes programas.

El concepto de herencia en POO proporciona una extensión importante a la idea de reutilización Un programador puede utilizar una clase existente, perteneciente a una biblioteca de clases y, sin modificarla, le puede añadir características y propiedades adicionales. Esto se hace por derivación de una nueva clase a partir de la existente. La nueva clase heredara las capacidades de la antigua, pero es libre de añadirle nuevas características a las suyas propias.

Por ejemplo, supongamos que se ha escrito o comprobado una clase que crea un sistema de menús, tal como los utilizados en Zortech 3.0, Turbo C++, Borland C++, etc. Desea añadir a las propiedades de la clase la capacidad de hacer que un menú parpadee o no a voluntad. Para hacer esta operación , cree simplemente una nueva clase que herede todas las propiedades de la clase existente, pero añadale las entradas pertinentes para obtener el parpadeode menú.

En síntesis, la creación de colecciones de clases fácilitara el desarrollo de nuevas aplicaciones a pertir de esas clases. Los programadores que trabajan en grupo pueden construir una aplicación como una serie de clases que son combinadas juntas para formar un programa final. También se puede utilizar y mejorar clases desarrolladas por otros miembros del proyecto sin hacer cambios en las realizaciones de las mismas.

# 16. SQLWINDOWS VS LENGUAJES DE POO

Naturalmente, la programación orientada a objetos requiere el uso de lenguajes de programación idoneos. Existen dos tipos de lenguajes POO: puros e híbridos.
Lenguajes puros son aquellos diseñados en base a las propiedades especificas de la programación orientada a objeto, y tienen unas construcciones lexicas y lógicas muy diferentes delas encontradas en los lenguajes estructurados tradicionales. Su primer representante fue Simula, un lenguaje de simulación basado en Algol; hoy dia, sin embargo, su representante más genuino es Smalltalk, y la versión Smalltalk/V la más difundida. Es el lenguaje al decir de los especialistas en POO, deberia ser elegido como primer lenguaje al comenzar a estudiar POO.

Los lenguajes híbridos son lenguajes que añaden, a las propiedades tradicionales estructuradas, caracteristicas orientadas a objetos.
Hoy dia son los más difundidos y, entre ellos, podemos citar: C++, Turbo Pascal (5.5, 6.0 y 7.0), Objetive Pascal y Objetive-C.

El debate sobre qué lenguaje utiizar es abierto y polémico. Los puristas sugieren lenguajes puros que fuerzan al programador a utilizar técnicas orientadas a objetos; los lenguajes híbridos permiten que otros paradigmas, como la programación estructurada, puedan ser utilizados.
La decision, muchas veces, dependerá de un conjunto de circunstancias impredecibles en el principio del diseño. Hoy en dia los lenguajes híbridos han proliferado debido a la gran comunidad existente de programadores Pascal y C, que logicamente tienen a priori asegurada una emigración más fácil a Turbo Pascal 5.5, 6.0 o 7.0 y a C++.

## 16.1 SMALLTALK

Es el primer lenguaje orientado a objetos. Descendiente de Simula-67, fue desarrollo en Xeros's Palo Alto Research Center (PARC). Smalltalk trata a todo como un objeto, y representa el paradigma de la programación orientada a objetos: es el lenguaje puro POO por excelencia.

El entorno Smalltalk es orientado a objetos, con ventanas, menús desplegables y emergentes, tratamientos de menús, etc. Presenta las caracteristicas fundamentales de POO, y cada objeto en el sistema Smalltalk hereda el objeto raiz. Al igual que otros lenguajes OO,
Smalltalk distingue entre objetos y clases. Un objeto es una instancia o ejemplo deua clase. Todas las variables instancia de una clase Smalltalk son por defecto, privadas a ese objeto.

Smalltalk es un lenguaje interpretado que le hace muy flexible; pero, lógicamente, limita sus posibilidades comerciales. Un programa Smalltalk debe ejecutarse desde dentro del entorno, y eso limita su transportabilidad. Esto no siempre es una desventaja. El entorno Smalltalk esta disponile en muchas plataformas.

## 16.2 TURBO/QUICK/OBJECTIVE PASCAL

En 1989, Borlan International y Microsoft, simultaneamente, lanzaron versiones orientadas a objetos de Pascal. Borland utilizo la filosofia C++, y Microsoft la filosofia Smalltalk, en sus versiones. Sin embargo, Borland ha progresado mucho y su ultima versión 7.0 incluye, además, una magnifica biblioteca de clases llamada Turbo Vision, mejora de las versiones anteriores5.5 y 6.0; por el contario, Microsoft sigue estancada en su versión primitiva.
Estos lenguajes soportan las caracteristicas minimas orientadas a objetos, como son la creación de clases, herencia (solo simple) y polimorfismo. La versión de Turbo Pascal es más robusta que Quick Pascal, aunque ésta es más fácil de aprender.

## 16.3 C++

C++, piedra angular de esta evaluación, es seguramente el lenguaje híbrido más popular, existen versiones para todas las plataformas: DOS, OS/2, Windows, UNIX, XENIX, etc. y muchos fabricantes comercializan productos y compiladores C++, como es el caso de AT&T, Borland, Microsot, Zortech, etc.
Su gran difusión actual y al que se preve en la decada de los noventa se debera esencialmente al proceso de estandarizacion que se encuentra sometido.
La versión 2.0 ya adoptada por el comite ANSI respectivo es seguida, practicamente, por todos los fabricantes. Las versiones 2.1 y 3.0 estan en proceso de estandarización y ya existen también actualizaciones de diversos fabricantes

## 16.4 TURBO C++ Y BORLAND C++

El compilador Turbo C++ es una implementación completa de AT&T C++ versión 2.0. Con Turbo C++ se obtienen todas las caracteristicas de C, asi como de C++. Su caracteristica fundamental es el entorno integrado de desarrollo, EID(IDE, Integrated Development Environment). En la actualidad, las versiones más populares son 1.0 y 2nd edition, aunque recientemente ha aparecido la nueva versión 3.0, que corre en entornso DOS y Windows.
Borland C++ es un paquete profesional. Añade las carateristicas básicas de los compiladores C y C++, un depurador incorporado, un profiler y un ensamblador. Otras caracteristicas sobresalientes son: soporte Windows 3.0/3.1 y modulos DLL/Dynamic Link Library, Biblioteca Dinamica de enlace. Además, Borland C++

contiene el Whitewater Groups's Resource Toolkit, que permite crear y mantener -programar- recursos Windows (iconos, menús, cuadros de diálogo, etc.). A finales de 1991, Borland ha presentado la versión 3.0, que soporta las versiones 2.0 y 2.1 de AT&T C++, y en la primera de 1992 la versión 3.1, que además soporta Windows 3.1.

## 16.5 MICROSOFT C++ 7.0

El progrma Microsoft C/C++ 7.0 (MSC) contiene un compilador C++ compatible con la versión C++ 2.1 de AT&T, así como un compilador C que sigue la norma ANSI. MSC necesita un PC basado en un procesador

## 16.6 MICROSOFT C/C++ 7.0

El programa Microsoft C/C++ 7.0 (MSC) contiene un compilador C++ compatible con la versión C++ 2.1 de AT&T, así como un compilador C que sigue la norma ANSI. MSC necesita un PC basado en un procesador 386 o 486 con un mínimo de 4MB de RAM, un disco duro de la menos 10MB librea, versión MS-DOS 3.3 o superior y monitores CGA, EGA, VGA o las nuevas SuperVGA(SVGA). MSC 7.0 esta diseñado para soportar aplicaciones DOS y Windows. El paquete contiene todo lo necesario para construir aplicaciones basadas en Windows: archivos de cabecera, bibliotecas, herramientas específicas, etc.

Una de las propiedades más notables de MSC 7.0 es la biblioteca de clases MFC (MICROSOFT FOUNDATION CLASSES), que ofrece soporte completo para objetos Windows tales como ventanas, diálogos, mensajes, fuentes, mapas de bits, etc. Esta biblioteca es una colección de 60 clases C++ reutilizables y que son compatibles totalmente con Windows 3.0 y 3.1. Una característica muy notable de esta biblioteca es su transaportabilidad, que no exige más que una recompilación de aplicaciones a otros entornos.

## 16.7 SQLWINDOWS

Explorando la programación Orientada a objeto. SQLWindows provee una completa implementación de un lenguaje de programación orientado a objeto.
Los programas orientados a objetos consisten de objetos de software independiente que tiene trabajos específicos y cada objeto es responsable de el trabajo que hace. En contraste, a los programas estructurados tradicionales, los objetos en OOP son autónomos. En general, para coger datos desde un objeto, usted envía un mensaje a el objeto o llama a alguna función definida en el objeto.

Adicionalmente con OOP en SQLWindows, usted crea código:

***Mas reutilizable:***
Si usted crea clases que oculten su implementación , usted será capaz de reutilizar el código en el mismo programa y otros programas.

***Mas confiable y fácil de mantener:***
El código reutilizable da como resultado programas más confiables y más fáciles de mantener

***Mas extendible***
Su código se hace más extendible por medio del uso del override a la clase derivada y las características de late-binding.

SQLWindows implementa estos conceptos de OOP:
Objetos
Clases
Herencia
Late-binding(polimorfismo)

Los objetos llevan los datos y el comportamiento que lo define. La parte de los datos de un objeto es igual que un registro o estructura en otro lenguaje. El comportamiento de un objeto toma la forma de funciones que accesan a esos datos. Usted puede escribir el código de un objeto de tal manera que no cuente con su implementación , ocultando información El objetivo del ocultamiento de información es hacer a un objeto independiente de otro tanto como sea posible Para usar el poder de la OOP, usted debe trabajar con clases. Una clase es el termino en OOP para tipo.

En los lenguajes procedurales, cada variable tiene un tipo, similarmente en OOP, cada objeto es miembro de una clase. Las clases en OOP son moldes (templates) usados para crear objetos similares con las mismas propiedades.

Usted puede definir sus propias clases que representen objetos visuales. SQLWindows también provee QuickObjects que son construidos mediante las características OOP de SQLWindows.

La herencia es la habilidad de definir clases en términos de otras clases. Nosotros veremos que la nueva clase es derivada de la clase vieja. La clase derivada automáticamente incluye todos los datos y funciones de la clase base. El SQLWindows provee de múltiple herencia, lo cual le permite a usted derivar otra clase de una o más clases.

El polimorfismo hace a su programa más flexible, usted puede llamar a una función miembro usando una variable que pueda ejecutarce varias veces en el programa, refiriéndose a diferentes objetos. En tiempo de ejecución , el sistema determinara la clase de el objeto y la función a ejecutar.

### 16.7.1 UN MEJOR MÉTODO

Usando OOP, usted puede crear un paradigma de objetos autónomos interactuando versus la programación tradicional estructurada procedural. EL modelo OOP usa los mismos conceptos durante la etapa de análisis y diseño en vez de una sintetización de un diseño para problemas de negocios resultando en varios procedimientos y estructuras de datos. Por ejemplo, con OOP, si usted necesita tratar una Orden de Compra como parte de un problema, usted puede tener un OBJETO DE ORDEN DE COMPRA en su diseño y trabajar con objetos de ordenes de compra.

### 16.7.2 DISEÑO DE CLASES FRECUENTEMENTE USADAS

Una manera de transformarse en un programador eficiente es identificar las piezas de programas que usted siempre necesita y crearlas como clases. Usted puede entonces usar estas clases durante el desarrollo de aplicaciones.

### 16.7.3 ESTANDARES CORPORATIVOS PARA INTERFACES DE USUARIOS

Uno de los casos en donde los departamentos MIS usan OOP esta en el esfuerzo de la creación de estándares, particularmente en las interfaces de los usuarios. Un buen ejemplo es definir una form windows class, el cual podría tener estándares
( por ejemplo: el color de fondo de la forma), logotipo de la compañía, y botones estándares sobre la barra de herramientas. Así todos las pantallas desarrolladas por el departamento de MIS estarán estandarizadas.

Otro, caso en donde la OOP es usada para estándares corporativos es en la de proveer una librería el clases para propósitos específicos. Una clase de data field no editable, con color de fondo amarillo, de tal manera que en el momento que un usuario vea un data field con estas características en la pantalla, el o ella

inmediatamente conozca que es un campo de datos de solo lectura. Similarmente esquemas de colores podría ser usados para diferenciar campos numéricos de campos string.

### 16.7.4 FÁCIL MANTENIMIENTO DEL CÓDIGO

Es muy buena idea tener siempre una base clase para cada objeto visual usado en una aplicación , aun si la clase base es la misma que provee el SQLWindows. más tarde será fácil replicar los cambios a través de todos los objetos con el solo cambio en la clase base.
Por ejemplo: proveer de un mecanismo de seguridad en las formas, detectando si no ha existido actividad en el teclado por más de 20 minutos, entonces la pantalla es destruida. Esta característica puede ser provista en la clase base, y automáticamente todas las formas derivadas la heredarán. Si se decide realizar un cambio 20 por 10, las formas derivadas lo detectarán automáticamente.

### 16.7.5 OCULTAMIENTO DE LOS DETALLES DE LA IMPEMENTACION

Una clase es esencialmente una unidad que contiene datos (variables) y algunos métodos (funciones manejadores de eventos). En un ambiente de orientación a objeto verdadero, los datos están completamente oculto de el mundo exterior. El mundo exterior puede manipular estos datos a través del uso de los métodos que provee el objeto. Los detalles internos son ocultos.