# UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

## "SISTEMA DE CAPTURA Y PRESENTACION GRAFICA BAJO AMBIENTE WINDOWS"

FALLA DE ORIGEN

**T E S I S**

QUE PARA OBTENER EL TITULO DE:

**INGENIERO EN COMPUTACION**

P R E S E N T A N :

**IGNACIO CABALLERO ROSAS**
**EDUARDO LOPEZ CASTELLANOS**


DIRECTOR DE TESIS: ING. GABRIEL CASTILLO HERNANDEZ

MEXICO, D. F. SEPTIEMBRE, 1995

TESIS CON
FALLA DE ORIGEN

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

**UNAM – Dirección General de Bibliotecas**
**Tesis Digitales**
**Restricciones de uso**

**DERECHOS RESERVADOS ©**
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

# AGRADECIMIENTOS.

**A LA UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO:**

Por su grandeza académica y forjadora de tan distinguidos profesionales que participan constantemente en el desarrollo del país.

**A NUESTROS PROFESORES:**

Por los conocimientos y experiencias que de ellos adquirimos día con día, contribuyendo con ello a nuestra formación profesional.

**A NUESTRO DIRECTOR DE TESIS**

Por otorgarnos su valioso y desinteresado apoyo en el desarrollo del presente trabajo.

**EN ESPECIAL A NUESTROS PADRES:**

Quienes nos guiaron a caminar por el sendero de la vida con pasos firmes.

**A NUESTROS HERMANOS:**

Por compartir con ustedes los momentos más difíciles durante la trayectoria de nuestra vida estudiantil.

**A NUESTROS AMIGOS:**

A quienes agradecemos la amistad fraternal que nos han brindado, esperando que la misma aflore cada vez más.

*Ignacio y Eduardo*

**Septiembre 1995.**

# INDICE GENERAL

# PREFACIO.

En la actualidad es difícil que el desarrollo de un sistema sea cumplido en el tiempo y el presupuesto original.

Peor aún, los sistemas que se liberan en estas condiciones están llenos de problemas y construídos de una manera muy rígida, volviéndose casi imposible hacer cambios importantes sin rediseñar totalmente el sistema.

La mayoría del software corporativo es obsoleto, aún antes de ser liberado y generalmente incapaz de evolucionar.

La capacidad de respuesta de los departamentos de cómputo, está pues, limitada por la complejidad, carga de trabajo y por la velocidad de cambio del negocio.

Esto indica que la técnica mediante la cual se están realizando los sistemas en la actualidad, no funciona de una manera adecuada.

¿No existirá acaso alguna forma mediante la cual se pueda crear un software rápido, barato, confiable y modificable?.

Es aquí donde entra la orientación a objetos, que plantea una nueva forma de desarrollar sistemas de cómputo, resolviendo algunos de los problemas en la construcción tradicional del software.

Con el fin de conocer los principales componentes de la orientación a objetos, sus ventajas y desventajas se elaboró esta tesis.

Asimismo, se desarrolló una aplicación (Sistema de Captura y Presentación Gráfica) con las técnicas orientadas a objetos con el propósito de aplicar los conocimientos adquiridos en la solución de un problema de la vida real.

La tesis consta de cinco capítulos.

En el capítulo uno, Introducción, se muestran los motivos por los cuales surge la orientación a objetos, se presentan las diferentes metodologías utilizadas en el desarrollo de sistemas.

El capítulo dos, Programación Orientada a Objetos, expone las características principales de un lenguaje orientado a objetos, beneficios y problemas potenciales.

En el capítulo tres, Análisis y Diseño Orientado a Objetos, se presentan los puntos básicos a considerar en el Análisis y Diseño de un sistema orientado a objetos.

El capítulo cuatro, Windows, muestra las causas por las cuales surge Windows, sus características y la Programación bajo ambiente Windows.

En el capítulo cinco, Análisis y Diseño Orientado a Objeto del Sistema, se presenta los pasos que se llevaron a cabo en el desarrollo del Sistema de Captura y Presentación Gráfica.

En las conclusiones exponemos los resultados de la investigación realizada, resaltando la importancia de un cambio en la forma en la cual se desarrollan los sistemas.

# CAPITULO 1. Introducción.

## 1.1 Crisis del Software.

La velocidad a la que avanza la tecnología del hardware de computadoras es sorprendente, año con año se logran avances que conducen a la construcción de computadoras más veloces, compactas y baratas.

Sin embargo, en el aspecto del software no existe un desarrollo tan acelerado como el del hardware, aunque los costos del hardware han disminuído, los costos del software han hecho lo contrario. Esta brecha entre el hardware y el software afecta a cualquier usuario de computadoras, aún a los más pequeños. Pero en las grandes industrias, estos efectos toman proporciones muy grandes, debido a la gran dependencia que tienen con sus sistemas de cómputo centrales, muchas veces obsoletos.

La construcción de software no es una tarea fácil y en muchas ocasiones los proyectos de programación sobrepasan los presupuestos de tiempo y costo, peor aun los sistemas que se liberan por lo general estan llenos de problemas y construídos de una manera muy rígida, en los cuales es casi imposible hacer cambios importantes. La mayoría del software corporativo es obsoleto, aún antes de ser liberado e incapaz de evolucionar. Muchas veces tarda menos tiempo el generar un nuevo sistema que modificar el ya existente.

Debe de existir alguna forma de producir un mejor software de una manera más rápida.

La construcción del software ha recibido la atención de los expertos desde hace mucho tiempo. Los enfoques de la programación han ido cambiando desde la invención de las computadoras para acomodarse a la crèciente complejidad de los programas. Desde la década de los 70's, se han desarrollado varias metodologías para el desarrollo de sistemas.

Destacan los trabajos del enfoque estructurado y el enfoque a datos.

El enfoque estructurado responde a las preguntas:
- ¿Cuáles son las entradas y salidas del proceso?
- ¿Cómo se transforman los datos?

Mientras que el enfoque a datos responde a:
- ¿Con qué elementos trabajo?
- ¿Cuál es su comportamiento?

Aunque estas metodologías han representado un gran avance y han sido efectivas durante largo tiempo, es evidente que la velocidad de cambio que requieren los negocios rebasan su capacidad.

Tomando las mejores ideas de estas metodologías, junto con otros nuevos conceptos, surge la Programación Orientada a Objetos (POO), con el fin de resolver estos problemas, dando una nueva visión de la tarea de la programación.

## 1.2 Metodologías utilizadas en el desarrollo de sistemas.

### • 1.2.1 Orientación a procedimientos.

Bajo este enfoque los sistemas se desarrollan utilizando módulos como el bloque fundamental de construcción.

Los datos se ven como entradas a una función de transformación para producir una salida.

La modularidad permite definir un sistema complejo en términos de unidades más pequeñas y manejables; cada una de esas unidades (o módulos) se encarga de manejar un aspecto local de todo el sistema, interactuando con otros módulos para cumplir con el objetivo global.

Las características principales que debe reunir un módulo son:

a) Un módulo debe corresponder con una unidad sintáctica del lenguaje (una subrutina,un paquete, una clase).

b) Los módulos deben de tener pocas interfases (medio de comunicación con otros módulos), y éstas deben ser pequeñas, con el fin de crear independencia entre módulos.

c) Cada módulo debe ocultar su implementación y algoritmos internos al resto del programa (principio de ocultamiento de información). No se debe permitir que un módulo modifique los elementos internos de otros módulos.

Algunas técnicas utilizadas bajo este enfoque son:
- Diagrama de Flujo de datos.
- Dependencia funcional.
- Jerarquía de funciones.
- Matrices.

Sobresalen los trabajos realizados para esta metodología de: Yourdon, Gane & Sarson, De Marco.

mantener el sistema son los mismos: identificación del cliente y estado de la cuenta.

Como podemos ver los datos son menos volátiles que los procedimientos.

- Los programas son diseñados para aplicaciones únicas.

- Los productos que resultan al emplear esta técnica son poco flexibles.

- No se alienta al programador a aprovechar el trabajo de proyectos anteriores.

- Alto nivel de redundancia.

- Para consultar información es necesario programar.

- Gran esfuerzo de los programadores y un alto riesgo humano.

Para solventar estos problemas surge la orientación a datos.

### 1.2.2 Orientación a datos.

Con el avance de las computadoras estos conceptos cambiaron, y se llegó a una nueva época dentro de la forma de almacenar y manejar la información de la Organización, independientemente de los procesos que la transformaran. Así nacieron los Sistemas de Información que son un método organizado para proporcionar información pasada, presente y que también permite obtener proyecciones para indicar comportamientos de las operaciones internas y externas de la empresa.

Estos sistemas de información estan constituídos por aplicaciones que generan información requerida para un componente específico de la organización, utilizando los datos del sistema.

A menudo diversas aplicaciones utilizan y modifican los mismos datos de la organización, asi que sería lo más indicado que todos se encontraran en el mismo lugar y guardaran integridad al ser modificados.

Se ve la necesidad de separar los datos de los programas, de manera que tengan la capacidad de interactuar, siendo independientes. El avance del Software, permite entonces a través de los nuevos dispositivos y productos para la administración de la información tener un control centralizado de la misma.

Esto se logra con el Concepto de Bases de Datos. Una base de datos es una colección de datos interrelacionados almacenados en conjunto, sin redundancias perjudiciales o innecesarias. Su finalidad es la de servir a una aplicación o más, de la mejor manera posible.

Características:

- Diseño lógico separado del diseño físico.
Se busca tener un esquema conceptual del sistema que describa las funciones y datos del sistema que son requeridos para operar el Negocio, independientemente del software y hardware donde estos se van a implantar.

- Los componentes de los procedimientos son derivados del flujo de datos.

- Se busca establecer Bases de Datos que controlen la integridad de los datos que manejan.

- Provocar que el usuario final sea el principal participante en el proceso de análisis.

El bloque de construcción principal bajo este enfoque son las estructuras de datos, esta importancia que se les da a los datos ayuda a los sistemas de información, dado que los datos son menos volátiles que los procedimientos.

La orientación a datos retoma los conceptos de la orientación a procedimientos, pero refuerza la parte débil de los datos: la integridad y la concurrencia.

Entre la BD física (almacenamiento real de los datos) y los usuarios del sistema existe un nivel de software, llamado manejador de BD, el cual maneja todas las solicitudes de acceso a la BD y se encarga de la integridad de los datos.

También forma parte de la Base de datos un lenguaje de definición de datos, en el que se definen las estructuras de los datos a utilizarse y un lenguaje de manipulación de datos en el que se mantienen las relaciones que existen entre los datos, además, permite al usuario accesar y manejar los datos.

Un sistema que se desarrolla en Bases de Datos, se fundamenta en un Modelo de datos. Gracias a este podemos identificar la estructura, contenido y relaciones de los datos que seran usados dentro del proyecto.

Es independiente del Hardware y Software con el que se desarrollará el sistema, y es representado gráficamente por el Diagrama Entidad-Relación.

Ventajas de utilizar Bases de Datos: Se reduce la redundancia, independencia de datos o programas, versatilidad, se pueden aplicar restricciones de seguridad, integridad de los datos, accesibilidad, compatibilidad, etc. Sobresale el trabajo de Jackson Michael en este enfoque.

Este enfoque representa una gran ayuda para los sistemas de información , pues incluye toda una metodología bien fundamentada con bases teóricas y que han sido objeto de constantes investigaciones.

Las ventajas de la orientación a datos se suman a las ventajas de la orientación a procedimientos.

Se evoluciona el concepto de la orientación a procedimientos hacia la orientación a datos.

No obstante, subsisten otros problemas:

- No existe una forma implícita en las Bases de Datos para promover el uso de una metodología.

  No es garantía el contar con una BD para resolver nuestros problemas, se depende de la experiencia y calidad de los diseñadores de la aplicación.

  Si no utilizamos correctamente la BD y no diseñamos bien nuestras relaciones entre los datos, resulta mejor no utilizar la BD.

- Otro gran problema es la dificultad de modificar la estructura de la BD, si se hacen cambios importantes se requiere restructurar, recompilar, readaptar la Base de Datos.

Para evitar esto, el concepto evoluciona a la orientación a Objetos.

### 1.2.3 Orientación a objetos.

A pesar de los esfuerzos que se han hecho para construir mejores sistemas, la crisis del software sigue creciendo año con año.

Han pasado mas de 40 años desde que se inventó la subrutina y se siguen construyendo los sistemas "a mano", una instrucción a la vez.

Se han desarrollado mejores métodos para este proceso de construcción que tratan de automatizarlo tales como: Pregeneradores de código, Interfases gráficas, SQL'S, 4GL'S, etc, pero no funcionan satisfactoriamente para sistemas grandes. Adicionalmente estos métodos dan como resultado un software que es dificil de mantener y modificar.[1]

Se requiere de una nueva metodología que permita desarrollar, depurar y mantener los sistemas más fácilmente, que pueda ser aplicable a sistemas grandes y chicos, que deje atrás los problemas tradicionales,. se necesita una manera mejor y más rápida de hacer sistemas. En lugar de construir los programas desde el principio cada vez, deberíamos de construirlos con piezas de software estándares y reutilizables.

---

[1] Object Oriented Technology: A Manager's Guide Taylor

La orientación a objetos puede cumplir con estos requerimientos.

**La orientación a procedimientos centra su atención en los procedimientos; La de datos en los datos; La orientación a objetos en AMBOS como un todo inseparable, ninguno es más importante que el otro.**

Se le llama orientación a objetos porque representa la abstracción (la capacidad para examinar algo, sin preocuparse por los detalles internos) de un objeto real, que consta de un estado, el cual es representado por los datos y cierto comportamiento que es representado por los métodos.

Así pues el objeto representa más fielmente al elemento de la realidad en la computadora; La computadora se amolda al mundo real, en contraste con la programación tradicional donde amoldamos el mundo con la computadora.

La orientación a objetos representa una evolución del concepto, pero una revolución del método.

Se evoluciona en el concepto porque se retoman los fundamentos de la orientación a procedimientos y de la orientación a datos. Un objeto consta de datos y métodos como un todo inseparable, ambos con la misma importancia. No es un concepto que salga de la nada.

Es una revolución del método porque nos obliga a seguir una cierta arquitectura, unas leyes básicas, que nos permitan garantizar un software más seguro, flexible, mantenible y reutilizable. Recordemos que en las otras orientaciones tenemos una arquitectura ( modularización, formas normales, etc) que

pueden aplicarse o no, quedando a la voluntad de los diseñadores.

Sin embargo, en la POO esta arquitectura esta implícita y no se le separa de la programación.[7]

De forma muy general, existen 3 partes medulares de la POO: Objetos, Mensajes y Clases.

**▸ 1.2.3.1 Objetos.**

Es una abstracción de una entidad tangible del mundo real o de un concepto teórico. Un objeto podría ser por ejemplo un avión, una puerta, una lista ligada, un trabajador, etc.

Un objeto se caracteriza por tener un cierto estado, comportamiento e identidad.

- El edo. de un objeto se modela por medio de valores variables que representan la abstracción del objeto y sus características (atributos), a lo que se conoce como datos.

- El comportamiento se modela a través de un conjunto de funciones que provocan la ejecución de alguna acción o el cambio del estado del objeto, a esto se le llama método.

- Un objeto tiene cierta identidad porque es único y diferente a cualquier otro.

[7] Alan Kay, MIT Media Lab "El Padre de la Computadora Personal"

### ▸ 1.2.3.2 Clase.

Es la agrupación de ciertos objetos que poseen datos y métodos comunes. Los objetos se crean mediante la instanciación de una clase. Mientras que un objeto es una entidad concreta que existe en el tiempo y el espacio, una clase representa solo una abstracción, la esencia del objeto.

Por ejemplo: dadas las variables x,z de tipo entero "int", decimos que x y z son objetos de la clase "int".

Una clase define las propiedades y métodos que comparten todas sus instancias, pero cada objeto tiene sus propios valores únicos.

La programación orientada a objetos propone dos estrategias para la reutilización de código: la composición y la herencia.

La composición permite definir una nueva clase de objetos mediante la unión de un conjunto de clases ya existentes.

Por ejemplo: la clase automóvil podria formarse mediante la unión de las clases: carroceria, llantas, motor, transmisión, etc.

Por otro lado, la herencia permite crear (derivar) una nueva clase basándose en otra clase más general. Una clase "derivada" adquiere todas las propiedades de la clase base. De esta forma puede ser posible derivar "pentágono" a partir de una clase "polígono".

#### ▸ 1.2.3.3 Mensajes.

Los objetos se comunican entre sí a través de mensajes. Los mensajes modifican el comportamiento del objeto receptor, provocando que haga cierta acción o que cambie su estado.

Por ejemplo: Preguntar por el valor de un dato, crear un nuevo objeto, modificar cierto estado de un objeto, etc.

Hay un tipo especial de mensajes que nos permiten manipular objetos de clases diferentes como si fueran de la misma clase (Polimorfismo), con lo cual es posible definir interfaces uniformes para diferentes tipos de objetos. Por ejemplo: Podríamos definir una función "suma" que actuara sobre valores enteros, flotantes, complejos, cadena de caracteres, etc.

En el siguiente capítulo se tratan con mayor profundidad estos puntos.

Con el propósito de mostrar la evolución de las metodologías en el desarrollo de sistemas, se presentó la clasificación anterior, sin embargo cabe aclarar que la POO surgió cronológicamente en paralelo con las otras orientaciones, enriqueciéndose con ellas.

## 1.3 Lenguajes orientados a objetos.

Algunas técnicas que son empleadas en la POO ya eran conocidas por los Ingenieros de Software desde hace algunos años, lenguajes como Ada o Modula-2 alentaban al programador a emplear alguna de ellas. Pero presentaban varios problemas:

Características como el manejo automático de memoria y la asociación dinámica de tipos imponen sobrecargas demasiado grandes a la ejecución de programas escritos en dichos lenguajes.

Hoy en día, se han venido solucionando estos problemas con el gran avance del Hardware, junto con ciertas estrategias que se han tomado para los lenguajes orientados a objetos.

Una de ellas consiste en utilizar lenguajes orientados a objetos para desarrollar los componentes de más alto nivel de un sistema y lenguajes funcionales para escribir las partes de bajo nivel críticas para la ejecución (como usar Smalltalk y C).

Otra estrategia es la de desarrollar nuevos lenguajes que nos proporcionen todas las facilidades de la POO, pero que no impongan sobrecarga de ejecución demasiado altas, a estos lenguajes se les llama *híbridos* tales como: C++, Turbo Pascal 5.5 de Borland.

Esto en conjunto con las ya mencionadas ventajas de la POO han provocado una gran aceptación de la POO en los diseñadores de sistemas.

Entre los lenguajes orientados a objetos mas populares, se encuentran Simula67, un lenguaje de simulación con facilidades para manipular eventos discretos; Smalltalk, usado para desarrollar interfases de usuario gráficas; Eiffel, que se aplica en diversas áreas.

# CAPITULO 2. Programación Orientada a Objetos.

La POO es más natural que la programación tradicional, ya que refleja las técnicas de la naturaleza para manejar la complejidad. Cuando se trata de construir sistemas complejos (seres vivos), la naturaleza no tiene igual. La naturaleza construye sistemas que, no importando lo complejo que sean, son muy flexibles para adaptarse a los cambios del medio ambiente y que en la mayoría de los casos también se autoreparan, el ser humano es una muestra palpable de esto, en él, interactúan entre sí una gran cantidad de sistemas (respiratorio, digestivo, circulatorio, nervioso, etc), además, el hombre posee la capacidad de adaptarse a diferentes medios ambientes, y si llega a sufrir algún daño, existen defensas propias para tratar de solucionar el mal.

La Orientación a Objetos es una nueva forma de construir sistemas de cómputo que resuelve algunos de los problemas principales en la construcción tradicional del software.

Una de las ideas principales es construir el software de la manera en la que construimos el hardware. En la programación, haciendo un símil con el hardware, nos encontramos como en la época en que se construían las computadoras con transistores.

En lugar de construir los programas desde el principo cada vez, deberíamos construirlos con piezas de software estándares y reutilizables; en el software, debemos pasar de los transistores a los circuitos integrados.[1]

---

[1] Philippe Kahn, Fundador de Borland Internacional

Dentro de la POO se manejan varios conceptos como: clase, polimorfismo, herencia, etc. Sin embargo, resulta confuso hablar de ellos separadamente, por lo que se les agrupa en 3 partes medulares:[4]

1) **Objetos.**
   Abstracción
   Encapsulado.

2) **Clases.**
   Herencia.
   Objetos compuestos.

3) **Mensajes.**
   Polimorfismo.
   Sobrecarga de funciones.
   Sobrecarga de operadores.
   Funciones Virtuales.

En muchas lecturas del tema, estos conceptos se muestran como partes sin mucha relación entre sí, lo que puede provocar confusión. La clasificación de arriba puede ser adecuada para comprender mas fácilmente los términos básicos de la POO.

### ▸ 2.1 Objetos.

Los Objetos representan una abstracción de algún elemento de la realidad. Los elementos de la realidad que representa el objeto puede provenir, principalmente de dos ramas:

---

[4] J.C. Cárdenas. Disertación sobre la POO.

1) Una cosa tangible o visible, tal como: una mesa, avión, un ferrocarril, etc.

2) Un concepto del pensamiento, un concepto generalizado, como una cola, una reacción química, un compilador, etc.

Como ya se mencionó, un objeto consta de datos internos que representan su estado, y de métodos que representan su comportamiento. Ambos, forma parte medular del objeto y no se puede pensar en ellos de forma separada.

El estado de un objeto se modela a través de un conjunto de valores variables, que representan la abstracción del objeto y sus características en el tiempo. A esto se le conoce como datos.

Asimismo, el comportamiento del objeto se modela a partir de un conjunto de funciones que llevan a cabo ciertas acciones o que modifican el estado del objeto. A esto le llamaremos método.

Existen dos conceptos asociados al objeto: Abstracción y Encapsulado.

### ▸ 2.1.1 Abstracción.

El concepto de Abstracción surge dentro de la orientación a procedimientos.

La abstracción es una descripción simplificada que resalta solamente las características esenciales de un objeto real, despreciando las características no esenciales.

En la orientación a procedimientos se abstrae las funciones que hace el sistema, pero los datos quedan en un segundo plano.

En la orientación a datos se modelan los objetos del mundo real en términos de sus datos; aunque los procedimientos son importantes bajo este enfoque, se tratan datos y procedimientos como partes separadas.

En la orientación a objetos, se abstrae el estado del objeto (en términos de sus datos) y de su comportamiento (método) como un todo inseparable, lo que lo hace más natural.

Por medio de la abstracción podemos representar los objetos de la realidad, resaltando sus características esenciales, pero, ¿Cuáles son esas características esenciales? bueno, eso depende del observador, de su experiencia y del punto de vista que utiliza para describir al objeto.

### ▸ 2.1.2 Encapsulado.

Es la forma de proteger a ciertos datos y ciertos métodos del objeto de la intromisión externa. Esto es, los usuarios del objeto NO PUEDEN (y no necesitan) accesar esos datos y métodos si el diseñador del objeto no lo permite.

Al proteger la implantación interna, simplificamos también su uso. Por ejemplo: un usuario de tarjeta de crédito no necesita saber los complicados procedimientos que tienen que llevarse a cabo para que él disponga de efectivo.

Este concepto ya existía en la orientación a procedimientos (independencia entre módulos). Sin embargo, los lenguajes de programación con orientación a procedimientos, no tienen un

mecanismo implícito para promover el ocultamiento de información, a diferencia de los lenguajes de programación orientados a objetos, donde esta aplicación está implícita en su arquitectura.

Ventajas de utilizar encapsulado:

1) Mejor manejo de la complejidad.
2) Los cambios a un objeto no afectan a los demás.
3) Aislar problemas de código.
4) Flexibilidad.

La abstracción y el encapsulado son conceptos complementarios. La abstracción permite a las personas modelar el problema con el que se enfrentan, mientras que el encapsulado permite la realización de cambios con relativamente poco esfuerzo.[5]

Para que funcione correctamente la abstracción, la implantación debe de estar encapsulada. De otra manera se dañaria la abstracción inicial cayendo en incosistencias.

Esto conduce a dos conceptos importantes:

**Cada objeto tiene dos partes importantes: interfase e implantación.**

La interfase del objeto es la vista externa, la forma en que lo ven los usuarios y está constituida por los métodos (públicos) del objeto.

---

[5] Booch , Grady Object Oriented Design with Applications.

La implantación del objeto son sus adentros, es la abstracción que el observador hace del elemento. Por lo tanto, debe de estar protegida del exterior para evitar inconsistencias; el modificar un dato puede requerir modificar otros para mantener la integridad.

## • 2.2 Clases.

Una clase es la definición abstracta de un grupo de objetos con datos y métodos comúnes. Los objetos se crean mediante la instanciación de una clase. Mientras que un objeto es una entidad concreta que existe en el tiempo y espacio, una clase representa solo una abstracción, la esencia del objeto.

Las clases definen las características comunes de un grupo de objetos. *Un objeto es la instancia de una clase*. Como tal, es una combinación de las características que comparten varios objetos, expresadas dentro de la clase, y de un estado único representado por los valores que tomen los datos de la instancia.

Una analogía con los lenguajes de programación orientados a procedimientos: decimos que la variable A es de tipo entero, entonces -en términos de la POO- el objeto A es una instancia de la clase entero.

Con las clases podemos definir nuevos tipos de variables.

Haciendo una comparación con los lenguajes de programación tradicional, una clase es como si definiéramos un nuevo tipo de variable preestablecida como son los enteros, char, etc. De esta forma podríamos tener variables tipo reloj, helicóptero, elevador, etc. El compilador los tratará como cualquier otra estructura del lenguaje ya establecida.

De hecho, la actividad principal de cualquier lenguaje de programación orientado a objetos es, precisamente la definición de nuevos tipos de variables, mediante la definición de sus datos y métodos.

A continuación se presenta un ejemplo de la declaración de una clase en C++:

```
class  linea
{
// Datos
     int color;
     int principio, final;
     int ancho;
// Métodos
public:
     int pon_color(int col);
     int define_linea(int x, int y);
     int cambia_ancho(int k);
}
```

Como podemos apreciar, la declaración de arriba se compone de: **datos**: color, principio, final, ancho; Y de **métodos**: pon_color(), define_linea(), cambia_ancho().

Las clases traen orden a los objetos organizándolos en jerarquias de clases: **Herencia y Objetos compuestos.** Estos conceptos se explican mas adelante, pero veamos cual es su origen.

**Métodos de Organización.** Para entender el mundo que nos rodea, las personas utilizamos varios métodos de organización.

De acuerdo a Yourdon y Coad, los métodos de organización que existen son tres:

1) Identidad.
2) Parte de.
3) Tipo de.

1) *Identidad.* La actividad principal en este punto es la de identificar los objetos por medio de sus atributos y comportamiento. Cada objeto es único y diferente a cualquier otro. Por ejemplo: Un avión, una bicicleta, un perro.

2) *Parte de.* Esta organización explica al objeto en término de sus componentes. A este tipo de objetos que se forman a partir de otros se les llama **objetos compuestos.** Casi cualquier objeto de la realidad esta compuesto por otros. Por ejemplo un coche consta de ruedas, motor, carrocería, asientos, etc.

3) *Tipo de.* Se describe al objeto en término de sus semejanzas con otros. Por ejemplo, para definir a un objeto "sillón", si conocemos lo que es una silla, podemos decir: un sillón es como una silla, solo que es más grande y cómodo.

**2.3 Herencia.**

Es una jerarquización que utiliza la organización *tipo de.* La herencia es la base principal en la reutilización de código dentro de la POO. Este mecanismo permite definir una clase añadiendo características (datos y métodos) a una clase ya existente. Una clase que es heredada por otra se llama *clase base* y la clase que hereda se llama *clase derivada o clase hijo.*

Dado que la clase hijo hereda los datos y métodos de la clase base, las jerarquizaciones siempre deben ser de lo general a lo particular.

En la herencia, las clases se definen como casos especiales de otras clases, formando así lo que llamamos una **jerarquía de clases.** La ventaja de definir las clases en una jerarquía es que, a través de la herencia, todos los casos especiales comparten las características de la clase base.

Por ejemplo, a continuación se muestra una clase llamada *vehículo_ruedas*, que define toscamente los vehículos que van por la carretera. Almacena el número de ruedas que tiene un vehículo y el número de pasajeros que puede transportar.

```
class vehiculo_ruedas
{
     int ruedas;
     int pasajeros;
public:
     void numero_de_ruedas(int num);
     int obten_numero_de_ruedas(void);
     void numero_de_pasajeros(int num);
     int obten_numero_de_pasajeros(void);
};
```

Podemos usar esta definición tosca de un vehículo de ruedas para definir objetos específicos. Por ejemplo, podríamos declarar una clase *camion* usando *vehiculo_ruedas*:

```
class camion : public vehiculo_ruedas
{
     int carga;
public:
     void capacidad_de_carga(int tamaño);
     int obten_capacidad_de_carga(void);
};
```

Los objetos que se declaren como instancias de la clase *camion* tendrán acceso a los métodos y datos de *vehiculo_ruedas* como si hubieran sido declaradas dentro de *camion*.

La herencia es un mecanismo muy eficiente, porque cada método y cada variable se define una sola vez, en el nivel más general que se aplique.

Al utilizar la herencia el software se vuelve más fácil de modificar. Se mejora la **productividad** en el desarrollo de software, se minimizan los cambios en el código, se reduce el tiempo de volver a probarlo. Dada la forma en que trabaja, los cambios en los datos y métodos de la clase "base" se propagan automáticamente a las clases "hijos".

La experiencia del diseñador y el contexto del sistema son decisivos en la estructura de la herencia.

### 2.3.1 Herencia múltiple.

Cuando una clase hijo tiene una sola clase base se le llama *herencia simple*. No obstante, es posible que una clase herede atributos de dos o más clases, a ésta se le conoce como *herencia múltiple*.

Aunque la herencia múltiple puede simplificar ciertas situaciones, también puede provocar ciertas complicaciones. Por ejemplo, supongamos que tenemos una clase hijo que hereda de dos clases bases, si ambas tienen un método que se llama igual, ¿Cuál de los métodos se heredará?.

Otro problema que trae la herencia múltiple es que muchas veces es mal utilizada. Para aplicarla correctamente, debemos estar seguros de que un objeto es el resultado de dos o más clases. Una regla sencilla para saber si una clase es o no heredable, es simplemente preguntar si la clase que queremos agregar es un **tipo de** la clase base de la que queremos heredar, si es así, procede la herencia.

Por ejemplo, un *camion* es un tipo de *vehiculo_ruedas*, pero una *llanta* **no es un tipo de** *vehiculo_ruedas*. Sin embargo, una *llanta* es una **parte de** *vehiculo_ruedas*. Veremos esta jerarquización en el siguiente punto.

### 2.3.2 Objetos Compuestos.

Como ya se vió, la jerarquización **tipo de** es representada por la **herencia.** Aunque la herencia es de gran ayuda dentro de la POO, no siempre es aplicable. Otra gran rama, es la jerarquización **parte de** que es referida por medio de **los objetos compuestos.**

Esta jerarquización es la que hacemos al distinguir a un objeto y las partes que lo componen. Por ejemplo, una casa consta de paredes, techo, piso...Estos a su vez constan de cemento, madera, clavos, etc. La composición permite definir una nueva clase de objetos mediante la unión de un conjunto de clases ya existentes.

Gran parte del poder de los **objetos**, consiste en que pueden contener también otros objetos. A los objetos que contienen otros objetos se les conoce como: **objetos compuestos.**

En la mayoría de los sistemas, los objetos compuestos no "contienen" literalmente otros objetos, sino que contienen datos que apuntan a una referencia a ese objeto.

Ventajas de usar Objetos Compuestos:

1) Los objetos *contenidos cambian más fácilmente*, tanto en tamaño como en composición, sin afectar la estructura de los objetos que los contienen. Esto hace que el mantenimiento de los sistemas, que utilizan este anidamiento, sea más fácil y rápido.

2) Los objetos *contenidos* pueden ser parte de cualquier número de objetos compuestos, en vez de estar ligados a un solo objeto, evitando asi la duplicidad.

Por ejemplo, supongamos que tenemos una clase llamada *industria* la cual requiere definir objetos que simulen procesos industriales. Una de las necesidades de esta clases consiste en proporcionar una forma de medir el tiempo, esta información puede ser proporcionada por una clase (ya definida) llamada *cronómetro*.

En vez de que la clase *industria* duplique la información del tiempo, solo hace una referencia a la forma de medir el tiempo.

Dado que la clase *industria* tiene solamente la referencia a la clase *cronómetro*, la clase *cronómetro* es libre de cambiar sin afectar a los objetos de la clase *industria*.

3) Los objetos *contenidos* en objetos compuestos pueden a su vez ser objetos compuestos, y este anidamiento puede llegar a cualquier nivel. La división en niveles facilita los cambios.

Podemos construir estructuras tan complejas como lo deseemos. La naturaleza maneja ampliamente esta técnica:

Un ser vivo no es un conjunto de células aisladas, sino que las células se organizan en unidades funcionales llamadas órganos, tales como el corazón y el cerebro. Los órganos a su vez se agrupan en sistemas, tales como el respiratorio y circulatorio. Finalmente estos sistemas constituyen un organismo que funciona como un todo. Los cuatro niveles de jerarquización de los organismos son muy convenientes para los biólogos, ya que traen **un orden** claro a lo que podría ser una colección caótica de interracciones entre células. De particular valor es que podamos entender un nivel sin tener que ver con los niveles alternos. Por ejemplo, podemos comprender el funcionamiento del cuerpo humano como una interacción entre sus sistemas, sin tener que entrar a detalle a nivel células. Entre más independientes sean los órganos entre si, más fácilmente podrán evolucionar individualmente sin afectar la operación de otros órganos y sistemas.

## ▸ 2.4 Mensajes.

Un mensaje es, simplemente, el requerimiento que hace un objeto a otro, para que el segundo ejecute uno de sus métodos.[6]

---

[6] Taylor, "Object Oriented Information Systems".

Por definición, al objeto que envía el requerimiento se le llama objeto *transmisor* y al objeto que lo recibe se le llama objeto *receptor*.

El conjunto de todos los métodos del objeto, determinan su comportamiento. Dado que el usuario del objeto sólo requiere conocer los métodos que necesita para utilizarlo, la interfase entre el objeto transmisor y receptor se encuentra bien definida.

Estructuralmente un mensaje consta de 3 partes:

1) La identidad del objeto receptor.
2) El nombre del método que desea ejecutar.
3) El conjunto de parámetros de ese método.

Para un mensaje, la secuencia de los eventos es la siguiente:

1) El transmisor envía el mensaje.
2) El receptor ejecuta el método, con los parámetros que le envía el transmisor.
3) El receptor regresa al transmisor una respuesta confirmando que recibió y ejecutó el mensaje.

**2.5 Polimorfismo.**

Dado que los objetos son independientes entre sí, es posible utilizar el mismo nombre de un método siempre y cuando se trate de objetos diferentes. A esto se le conoce con el nombre de **Polimorfismo.**

El Polimorfismo es el proceso por el cual se pueden acceder a diferentes implementaciones de una función, con el mismo nombre. Por esta razón, a veces el polimorfismo se caracteriza por la frase "*Una interfaz, múltiples métodos* ".

Permite usar un nombre para varios propósitos relacionados, pero ligeramente diferentes. Su fin es permitir el uso de un nombre para especificar una clase de acción general. Se ejecuta una parte específica de la clase general dependiendo del tipo de dato con el que se este tratando.[7]

"El Polimorfismo se refiere a que cada objeto puede reaccionar de manera diferente a un mismo mensaje".[8]

Veamos un ejemplo:

Supongamos que nos interesa tener funciones que nos permitan realizar la suma de enteros, la suma de complejos, concatenar cadena de caracteres.

En la programación tradicional debemos declarar funciones con nombres diferentes para cada labor:

```
suma_entero(3,4)
suma_complejo(3+4i,6+9i)
suma_cadenas("programación","objetos")
```

Lo anterior, genera una explosión en el número de funciones a manejar, haciendo que crezca la complejidad. En la POO podemos

---

[7] Schildt, "Turbo C++".

[8] Cárdenas, "Disertación sobre la POO".

utilizar el mismo nombre: suma(), sin tener que renombrar cada función.

Reescribiendo las funciones:

```
suma(3,4)
suma(3+4i,6+9i)
suma("programación","objetos")
```

El compilador escoge la función que deseamos, dependiendo del parámetro que estemos utilizando. El concepto de "sumar" es el mismo, solo que su manejo es diferente. Este es el concepto del Polimorfismo.

Formas de conseguir el Polimorfismo:

- Sobrecarga de funciones.
- Funciones Virtuales.
- Sobrecarga de operadores.

## 2.6 Sobrecarga de funciones.

Dos o más funciones pueden compartir el mismo nombre siempre que su declaración de parámetros sea diferente. En esta situación, las funciones que comparten el mismo nombre se dice que están sobrecargadas. La sobrecarga de funciones ayuda a gestionar la complejidad.

Ejemplo:

```
void print(char *);
void print(int);
main( )
{
     print(3);
     print("\n Hola");
}

void print(int n)
{
     cout << "Dentro de la función print que utiliza un ";
     cout << "argumento entero";
     cout << "el numero fue " << n;
}

void print(char *cadena)
{
     cout << "Dentro de la función print que utiliza una ";
     cout << "cadena de caracteres";
     cout << " la cadena fue " <<  cadena;
}
```

## 2.7 Funciones Virtuales.

Es una forma especial de sobrecarga de funciones. Cuando se sobrecargan funciones normales, el tipo devuelto, el número y los parámetros requeridos por la función pueden variar. Pero, en las funciones virtuales los prototipos deben de coincidir.

Una función virtual es una función que se declara en la clase base como virtual y se redefine en una o mas clases derivadas.

Las funciones virtuales son especiales porque cuando se accede a una de ellas, usando un apuntador de la clase base a un objeto de la clase derivada, se determina en tiempo de *ejecución* a que función llamar, en función del tipo de objeto apuntado.

Las funciones virtuales permiten a una clase general (base) especificar que funciones seran comúnes a cualquier clase derivada de ella, aunque permitiendo a las clases derivadas especificar la implentación de alguna o de todas estas funciones. En otras palabras, la clase base dicta la *interfaz* general que tendrá cualquier objeto de una clase derivada, pero deja a la clase derivada definir el método *real*.

Veamos un ejemplo:

```
class figura     /* definición de la clase Base */
{
     int x, y;
public:
     void dimensiones(int i, int j)
     {
          x= i;
          y= j;
     }
     virtual void muestra_area(void)
     {
          cout << "No se define cálculo de area";
          cout << " Para esta clase.\n";
     }
};
```

```
class triangulo:public figura     /* clase derivada de figura */
{
public:
     void muestra_area(void)
     {
          cout << "Triangulo con altura ";
          cout << x << "y base" <<  y;
          cout << "tiene un área de ";
          cout << x * 0.5 * y << "\n";
     }
};

class rectangulo:public figura   /* clase derivada de figura */
{
public:
     void muestra_area(void)
     {
          cout << "Rectángulo con dimensiones ";
          cout << "largo" << x << "ancho" << y;
          cout << "tiene un área de ";
          cout << x * y << "\n";
     }
};

main(void)
{
     figura *p; /* se crea un apuntador a la clase base */
     triangulo t; /* se crea un objeto triángulo */
     rectangulo r; /* se crea un objeto rectángulo */

     p=&t;  /* apuntador de la clase base a un objeto (triángulo)
            de la clase derivada */
     p->dimensiones(10,5);
```

```
    p->muestra_area( );

    p->&r;     /*apuntador de la clase base a un objeto
                (rectángulo) de la clase derivada */
    p->dimensiones(3,4);
    p->muestra_area( );

    return 0;
}
```

Como se puede apreciar en este ejemplo, la interfaz con *rectangulo* y *triangulo* es la misma, aunque ambas proporcionen sus propios métodos para calcular el área de cada uno de sus objetos.

**2.8 Sobrecarga de operadores.**

Otra forma de conseguir el polimorfismo es a través de la *sobrecarga de operadores*.

A la mayoría de los operadores de los lenguajes que son orientados a objetos se les puede dar significados especiales con relación a clases específicas. Cuando se sobrecarga un operador no se pierde ninguno de sus significados previos, simplemente significa que se ha definido una nueva operación con relación a una clase específica.

Se debe crear una clase que defina el tipo de datos sobre los que trabajará el operador sobrecargado.

En el siguiente ejemplo se sobrecarga el operador ^ de modo que pueda ser usado para elevar un número a una potencia. Como 10^2 sería = 100.

Primero se debe construir una clase que defina el tipo de datos que utilizarán el operador sobrecargado ^

```
class pot
{
public:
      int num;
      friend int operator^(pot b, pot e);
};

pot b,e;

void main(void)
{
   b.num= 2;
   e.num= 4;
   printf("%d",b^e);
}

// operacion x^y
int operator^(pot b,pot e)
{
      int t,temp;
      temp= b.num;
      for(t= e.num-1; t; t--) b.num *= temp;
      return b.num;
}
```

Ahora, siempre que se utilicen variables de tipo pot en una expresión de la forma b^e, se obtendrá la potencia en vez de la función de bits XOR. Recordemos que la operación XOR sigue existiendo, pero no para los datos de tipo pot.

## 2.9 Conclusiones de la POO.

Debemos estar conscientes que el cambiar a esta nueva tecnología traerá como consecuencia nuevas ventajas y problemas. Todo parece indicar que los beneficios sobrepasan considerablemente a los problemas de la migración.

### 2.9.1 Beneficios Potenciales.

**1) Desarrollo más rápido.**

Los Sistemas Orientados a Objetos se desarrollan eficazmente, debido a la reutilización de objetos y programas existentes.

Si no se cuenta con una infraestructura en bibliotecas de clase, programar un sistema desde lo más básico llevará más tiempo que desarrollarlo con los métodos tradicionales. Afortunadamente, existen comercialmente muchas bibliotecas disponibles.

**2) Mantenimiento más fácil.**

El uso de herramientas de la POO tales como: herencia, polimorfismo, objetos compuestos, etc, constituyen el medio ideal para crear sistemas fácilmente modificables.

Este punto es de suma importancia, especialmente si tomamos en cuenta que:

- El 80% del personal de los departamentos de cómputo se dedican al mantenimiento de programas.

▸ Los gastos que se efectúan para hacer mantenimiento al sistema, representan alrededor del 70% de su costo total.[9]

▸ La vida útil de un producto de software puede ser cinco o seis veces más grande que en el lapso en el que se desarrolla.

**3) Flexibilidad.**

En el mundo actual las necesidades del usuario cambian dinámicamente, es por ello que se requiere sistemas adaptables, esto es, que tengan la facilidad de modificarse para realizar nuevas funciones.

Aproximadamente el 60% de los costos de mantenimiento de un sistema (alrededor del 42% del costo total) se tienen que hacer por cambios en las especificaciones del usuario o en los formatos de los datos.[10]

No es de extrañar que antes de liberar un Sistema, éste se encuentra ya obsoleto.

La Orientación a Objetos presenta alternativas para resolver esta problemática:
▸ La Escalabilidad. Permite la habilidad de construir sobre lo existente sin perturbarlo (herencia) de manera que la funcionalidad del sistema va evolucionando poco a poco.

▸ Estructuras más completas, estructuras que además de representar datos hacen operaciones sobre ellos (como una base de datos orientada a objetos).

---

[9] Arrieta Norberto, POO usando C++.

[10] Arrieta Norberto, POO usando C++.

▸ Adaptabilidad. Cubrir las necesidades del usuario y no las condiciones superpuestas por el diseñador original.

**4) Reduce costos.**

Si podemos desarrollar un software más rápido (punto 1), emplear menos recursos en mantenimiento, aprovechar bibliotecas ya desarrolladas, podremos abatir costos en el desarrollo de sistemas.

**5) Mejor calidad.**

Dado que los objetos a partir de los cuales se produce un nuevo software ya están probados, no tenemos la necesidad de verificar de nuevo toda la jerarquía, solamente utilizamos el objeto como tal o heredamos de él algunas características, para adicionar otras nuevas. De manera que lo que existía sigue funcionando igual y sólo se debera probar las adiciones efectúadas.

### 2.9.2 Problemas Potenciales.

**1) Disponibilidad de personal capacitado.**

Es necesario enseñar a nuestros desarrolladores en esta nueva tecnología, un problema al cual nos enfrentaremos es que la curva de aprendizaje es prolongada, típicamente es de 8 meses o más.

**2) Costos de conversión.**

El cambiar a un esquema de Orientación a Objetos implica una fuerte inversión en herramientas, educación y consultoría. Es por ello necesario planear bien la forma en la cual se efectuará el cambio según convenga a los intereses de cada empresa.

**3) Velocidad de ejecución.**

La mayoría de los lenguajes Orientados a Objetos imponen una carga bastante pesada a la computadora.

**4) Mala elección de tecnología.**

Si no tenemos las herramientas adecuadas (por ejemplo: compiladores, o lenguajes deficientes) nos crearemos un problema grande. Debemos llevar a cabo un análisis de los productos del mercado para ver cuales son los que satisfacen nuestras necesidades.

**5) Necesidad de estándares.**

Dado que muchas compañías quieren imponer sus estándares, no existen mas que tendencias en el mercado.

**6) Requerimiento de mejores herramientas.**

Hoy por hoy, es necesario que hagamos mucho trabajo manual, desde el análisis y el diseño (como sucedió con la metodología CASE) hasta las bases de datos orientadas a objetos.

### 2.9.3 Futuro.

El cambiar a un esquema Orientado a Objetos implica una planeación minuciosa y de la ayuda de consultores reconocidos. Si no construimos sólidamente nuestras bases en esta tecnología iremos a un fracaso inminente. Por el contrario, si logramos avanzar correctamente en este campo obtendremos una ventaja competitiva, mejorará nuestra productividad.

No cabe duda que el mercado se dirige rápidamente hacia la Orientación a Objetos. Grandes compañías como IBM, AT&T, Microsoft, etc. están apoyando fuertemente este enfoque.

# CAPITULO 3. Análisis y Diseño Orientado a Objetos.

## 3.1 Modelos de la Realidad.

Cuando se analizan sistemas, se crean modelos del área de la aplicación que nos interesa. El modelo representa un aspecto de la realidad y es una forma de ayudar a entenderla.

Con el análisis orientado a objetos, la forma en la cual se modela a la realidad difiere del análisis convencional. Se representa al mundo real en términos de tipos de objetos y qué ocurre con ellos. La realidad, despues de todo, consiste en objetos y eventos, que cambian el estado a estos objetos.

En la metodología tradicional de desarrollo, el modelo usado por el analista difiere del usado por el diseñador, el programador añade un tercer enfoque de la situación. Los Analistas usan Modelos de Entidad-Relación, descomposición funcional, matrices, etc, Los Diseñadores utilizan diagramas de flujos de datos, diagramas de acción, mientras que los programadores usan las instrucciones propias de algún lenguaje tal como C o Cobol.

Una y otra vez oímos decir: "Se creó un sistema técnicamente excelente, pero no era lo que el usuario deseaba". ¿Por qué habrá sido esto?, ¿No existirá acaso, algún medio mediante el cual Analistas, Diseñadores y Programadores junto con el Usuario puedan trabajar sobre una misma cosa?

Mediante el uso de las técnicas de la Programación Orientada a Objetos, Analistas, Diseñadores, Programadores y Usuarios, piensan sobre un mismo modelo. Todos ellos piensan en tipos de objetos y cómo se comportan.

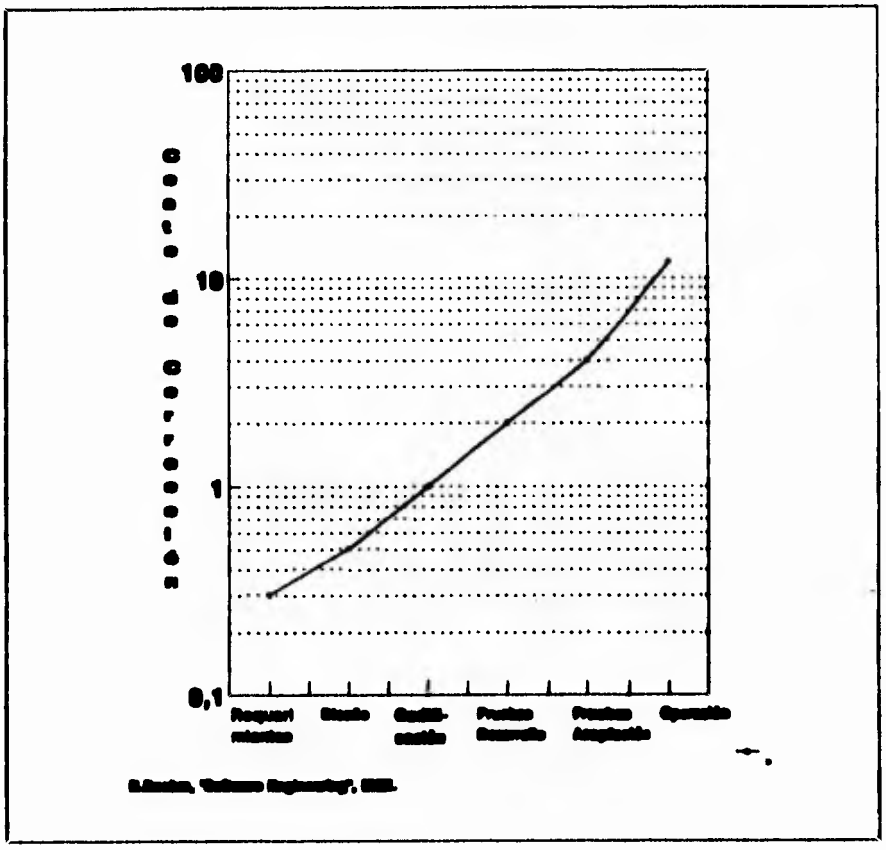
La transición entre el análisis y el diseño es más natural, se genera una como consecuencia de la otra, por lo tanto muchas veces resulta dificil señalar donde termina una y comienza la otra.

El uso de un mismo modelo trae como consecuencias:

- Mayor Productividad.
- Mejor comunicación entre Usuarios, Analistas, Diseñadores y Programadores.
- Mayor calidad en el Sistema.
- Más flexibilidad al cambio.
- Menor costo en el desarrollo del producto.
- Menos errores en el Sistema.

Barry Boehm[11] ha producido cierta evidencia manifiesta que muestra cómo el costo de corrección de un error crece fuera de toda proporción cuanto más tarde se detecta, en el ciclo de vida del sistema desarrollado.

[11] B. Boehm "Software Engineering", IEEE Transactions on Computers.




En la gráfica se puede observar el costo relativo de corregir un error -¡¡Nótese la escala logarítmica!!- dependiendo de la fase de desarrollo del sistema en que el error es detectado.

Así, un error que se advierte en la fase de requerimientos -debido tal vez a que el usuario examinó nuestro modelo lógico gráfico- puede costar 0.2 unidades (digamos $20), y en cambio si el mismo error no es detectado hasta que el sistema entra en operación, el costo resulta mayor a 10 unidades ($1000). Así, la construcción de un modelo que comunique al usuario qué es lo que puede y no puede hacer el sistema, es un ejercicio crucialmente importante en función del costo de corregir los errores más tarde. Realizar los cambios en una hoja de papel es barato: efectuar cambios en la codificación es muchas veces más caro, y realizar cambios en un sistema funcionando es muchísimo más caro aún. No es conveniente esperar a que el usuario "vea lo que recibe" antes de que "sepa lo que quiere".

### 3.2 Dos tipos de Modelos.

En el Análisis Orientado a Objetos[12], se construyen 2 tipos de modelos interelacionados entre si para representar al mundo:

Un modelo para representar a los tipos de objetos y su estructura. Aquí identificamos tipos de objetos, clases, relaciones entre objetos, herencia y composición. De esto se encarga el **Análisis Estructurado de Objetos (OSA) y el Diseño Estructurado de Objetos (OSD).**

Y otro modelo para ver que ocurre con estos objetos. Este aspecto concierne al comportamiento de los objetos y qué ocurre con ellos en el tiempo, esto es lo que trata **el Análisis del Comportamiento de los Objetos (OBA) y el Diseño del Comportamiento de los Objetos (OBD).**

---

[12] Según James Martin y James J. Odell.

Los modelos se representan mediante diagramas llamados esquemas. Los esquemas de objetos muestran la estructura del objeto y los esquemas de eventos muestran lo que le ocurre a los objetos.

**ANALISIS DE LA ESTRUCTURA DE UN OBJETO.**

- Tipos de Objetos.
- Asociaciones entre objetos.
- Generalización.
- Composición.

**ANALISIS DEL COMPORTAMIENTO DE UN OBJETO.**

- Cambios del Objeto en el tiempo.
- Tipos de Eventos.
- Estados de un Objeto.
- Reglas de disparo de un evento.

**DISEÑO DE LA ESTRUCTURA DE UN OBJETO.**

- Clases.
- Superclases y Subclases.
- Herencia.
- Estructuras de Datos.

**DISEÑO DEL COMPORTAMIENTO DE UN OBJETO.**

- Métodos y Operaciones.
- Lógica Procedural.
- Diseño de pantallas.
- Prototipos.

En el **Análisis de la estructura de un objeto** nosotros definimos las categorías de objetos y la forma en la cual ellos se asocian.

Del resultado de esto, en el **Diseño de la Estructura de un Objeto**, nosotros identificamos a las clases (la implantación de los tipos de objetos), superclases, subclases, definimos estructuras de datos.

En el **Análisis del comportamiento de un objeto** modelamos qué ocurre con el objeto al transcurrir el tiempo. Definimos los tipos de estados que puede tener un objeto, identificamos los eventos que cambian estos estados, reconocemos la sucesión de estos eventos. Del resultado de esto en **el Diseño del Comportamiento de un Objeto** nosotros definimos los métodos, la lógica procedural, las pantallas, la interacción con el usuario y los prototipos.

### 3.3 Análisis de la Estructura de un Objeto.

Aquí definimos la tipos de objetos que nosostros percibimos y la forma en la cual ellos estan asociados.

Recordemos que un objeto es cualquier cosa real o abstracta acerca de la cual queramos almacenar datos y métodos que manipulen esos datos. Además un tipo de objeto es una categoria de objeto. Por ejemplo: El tipo de objetos "Trabajador" se aplica a aquellos objetos que laboran en alguna compañía. Instancias de "Trabajador" podrían ser Luis, Pedro, etc.

Un objeto es una instancia de un tipo de objetos.

Durante esta fase se deberán identificar los tipos de objetos más que identificar a los objetos individuales de un sistema.

Estos tipos de objetos guiarán al diseñador en la definición de clases y su estructura. Un objeto solo puede ser manipulado via las operaciones asociadas a su tipo. Sin tipos de objetos las operaciones no pueden ser definidas adecuadamente. Por ejemplo: operaciones tales como: Jubilación, Despido, Promoción están íntimamente ligados con el tipo de objetos "Trabajador", porque

ellas cambian el estado de un trabajador.

El tipo de objetos que nosotros definimos y usamos puede ser variado, porque nosotros los escogemos basados en la forma de como entendemos al mundo.

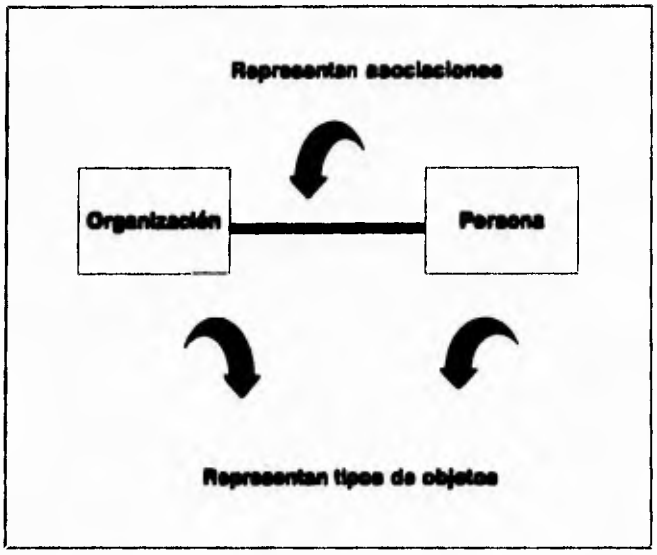
En efecto, un objeto puede ser categorizado en más de una forma. Por ejemplo una persona puede considerar al **objeto Juan** como **Hombre**. Su Jefe lo puede considerar como **Empleado**, el veterinario lo puede considerar como el **Dueño de lassie**, la tienda donde compra lo considera **Un cliente**.

### 3.3.1 Asociaciones entre objetos.

Otro punto importante a considerar es modelar la forma en la cual los objetos se asocian con otros objetos. Por ejemplo objetos **Organización** tales como IBM, DEC, NASA estan asociados con objetos **Personas** tales como Susana, Roberto.

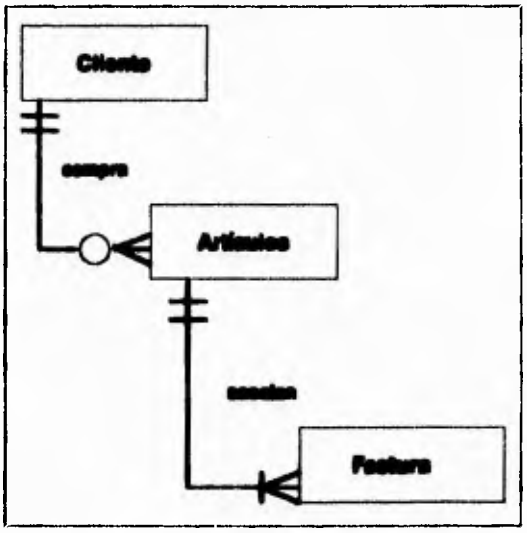
En el análisis, nombrar las asociaciones e indicar como los objetos de un tipo se asocian con objetos de otro tipo proporcionan una forma clara de entender al mundo.

Las asociaciones son representadas en un diagrama de la siguiente forma:




Para aumentar la comprensión de la asociación es bueno señalar su significado e indicar la cantidad de objetos con los que un objeto dado puede y debe asociarse.

Veamos otro ejemplo en donde se indica el tipo de asociación de un objeto con otro:



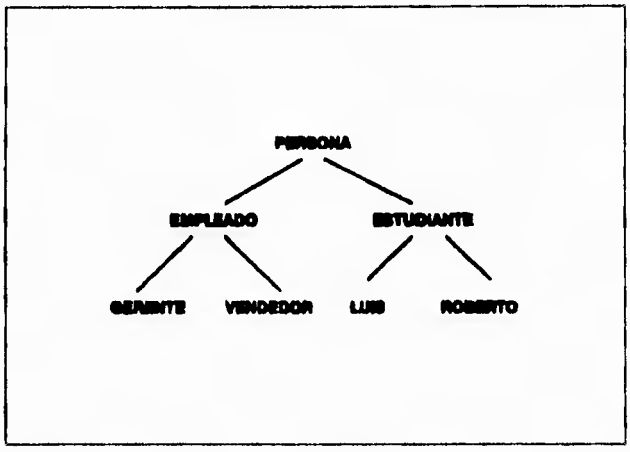

Un pedido es ordenado por solo un cliente. Un cliente compra desde 0 hasta muchos artículos. Una factura es asociada desde uno a varios artículos.

### 3.3.2 Generalizaciones.

Una de las formas en las cuales los seres humanos organizamos al mundo que nos rodea[13] es organizándolo en jerarquías que van de lo más general a lo más específico.

[13] Para una mayor referencia vease el Cap.2

Por ejemplo:




El tipo de objeto **Persona** es más general que el tipo de objeto **Empleado y Estudiante.** Esto implica que **Empleado y Estudiante** son **subtipos** de **Persona** o dicho de otra forma, **Persona** es un **supertipo** de Empleado y Estudiante.

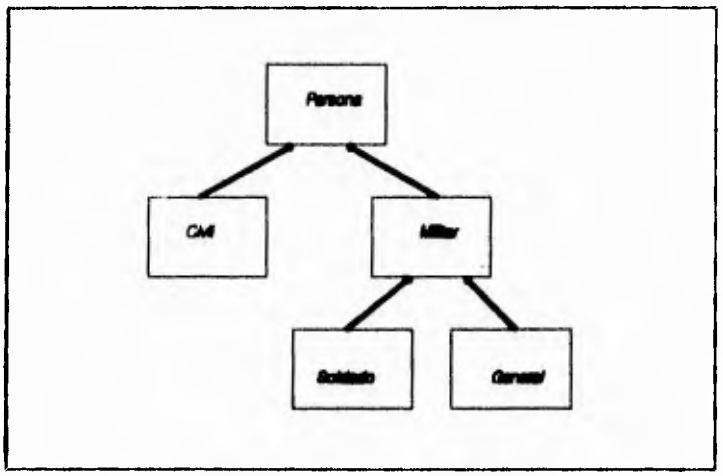
Todas las propiedades de un tipo de objeto son aplicadas a los subtipos. Por ejemplo todas los datos y métodos de un **Vehículo** son aplicadas al tipo de objeto **Ford.**

La Generalización es importante en la POO por dos razones:

▸ Usando la notación de lo subtipos y supertipos obtenemos una herramienta poderosa con la cual podemos describir el mundo de la aplicación.

• Esta indicará la forma en la cual debe de ser la herencia de las clases que posteriormente se definirán.

**Diagrama de Generalización:**

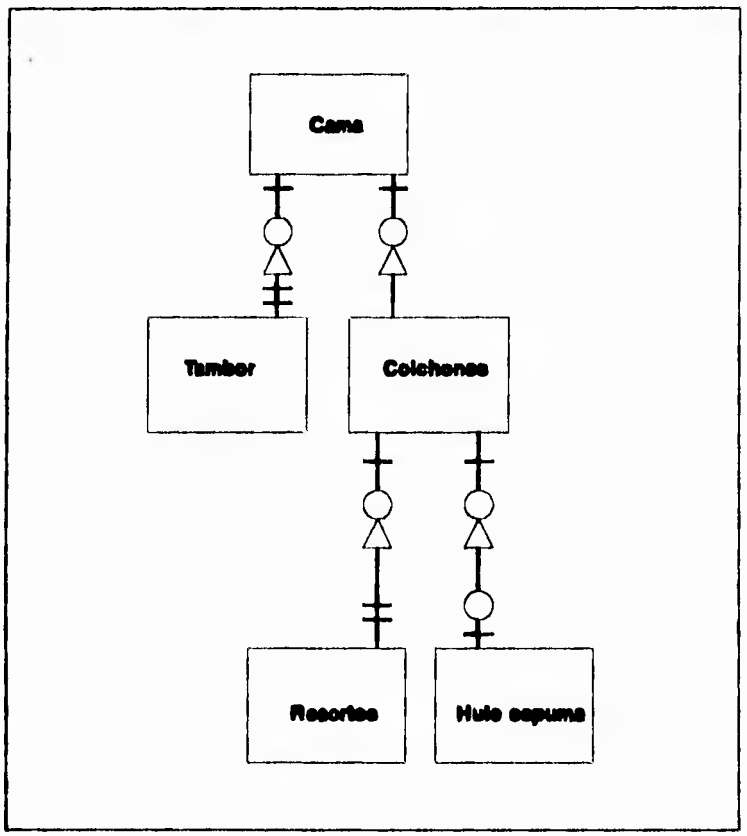


### 3.3.3 Composiciones.

Algunos tipos de objetos son descritos como "complejos". Entendiendo por "complejos" aquellos objetos que están constituidos por otros objetos. Por ejemplo: Un coche está constituido por llantas, carroceria, motor, etc.

La composición nos ayudará a determinar la estructura que deberá tener la(s) clase(s) al momento de su definición.

Diagrama de composición:




En este ejemplo se ve que una cama está constituída por un colchón y uno o más tambores. Cada colchón está compuesto de resortes y puede o no tener hule espuma.

**Esquema de Objetos.**

La comprensión de un modelo suele ser más sencilla si los tipos de objetos y relaciones se representan mediante un diagrama de relación entre objetos, los supertipos y subtipos se representan en un diagrama de generalización y las estructuras compuestas en un diagrama de composición. Al conjunto de estos 3 diagramas se le conoce con el nombre de **Esquemas de Objetos.**

## 3.4 Análisis del comportamiento de un objeto.

En esta etapa se ve qué ocurre con el objeto al transcurrir el tiempo, se definen los estados que puede tener, se identifican los eventos que cambian estos estados, se reconoce la sucesión de eventos.

### 3.4.1 Estados de un Objeto.

El estado de un objeto es la colección de los tipos de objetos que se aplican a él.

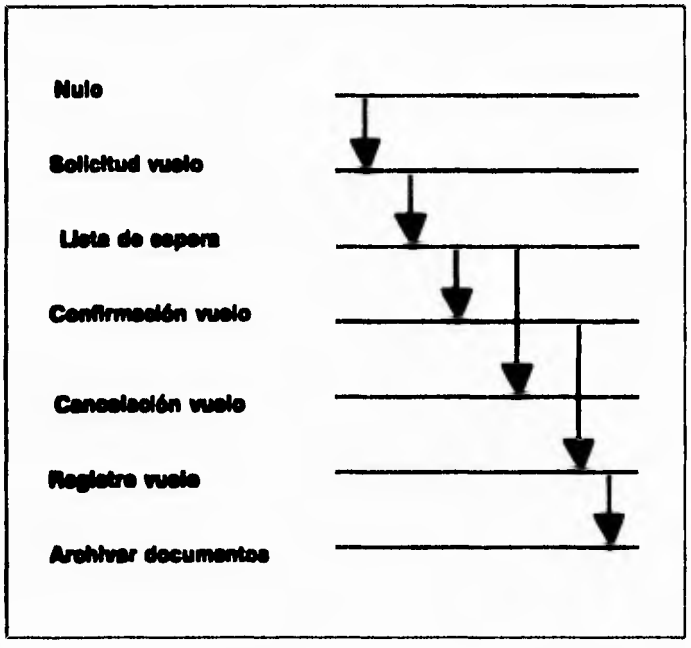
Un objeto puede existir en diferentes estados. Por ejemplo, supongamos que tenemos un objeto "Reservación Aerea", el cual puede ser una instancia de alguno de los siguientes tipos de objetos:

- Reservación solicitada.
- Reservación en lista de espera.
- Reservación confirmada de vuelo.
- Reservación cancelada.
- Reservación satisfecha (una vez que el avión ha despegado).
- Reservación archivada.

Al ocurrir un cambio de estado, el objeto pasa de una categoría a otra.

A los distintos estados que puede tomar un objeto se le conoce como Ciclo de vida de un objeto.

Veamos el Ciclo de Vida de un objeto "Reservación Aérea".




Las líneas horizontales representan los estados de la Reservación Aérea, las líneas verticales representan transición entre estados. A esto se le llama **Diagrama de transición de estados de un objeto.**

Sin embargo, un objeto puede tener muchas perspectivas del ciclo de vida. Por ejemplo, el mismo objeto "Reservación de Avión" podría tener estados relacionados con el pago, tales como:

- Depósito para el vuelo.
- Crédito para el vuelo.
- Devolución del costo del vuelo.

En un mismo momento este objeto podría tener el estado de confirmación de vuelo y depósito para el vuelo. En otras palabras, un objeto puede ser una instancia de varios tipos de objetos simultáneamente.

Al usar un lenguaje Orientado a Objetos, el estado está presente en los datos que se almacenan del objeto.

### 3.4.2 Eventos.

Nuestro mundo está lleno de eventos: Un cliente solicita una reservación de vuelo a una Aerolínea, un trabajador es ascendido de puesto, un coche choca contra otro, etc.

En el análisis orientado a objetos, el mundo es descrito en términos de objetos y estados, y los eventos cambian estos estados. Un evento es un cambio en el estado de un objeto. Sin eventos el mundo permanecería estático. Por ejemplo, cuando un empleado deposita dinero en su cuenta bancaria, ésta debe ser actualizada (debe cambiar del estado en el que se encuentra).

Básicamente, un evento describe los siguientes cambios de estado de un objeto:

- Un objeto es creado. Ejemplo: Se lanza un nuevo perfume al mercado.
- Un objeto es terminado. Ejemplo: Terminó la garantía de una PC.
- Un objeto es clasificado como una instancia de un tipo de objeto. Por ejemplo: Una esposa que llega a ser madre. Un empleado que llega a ser gerente.
- Un objeto es desclasificado como una instancia de un tipo de objeto. Ejemplo: Un banco que suspende a un cliente moroso.
- Un objeto cambia de clasificación. Por ejemplo: Un cuentahabiente cambia de una cuenta de ahorros a una cuenta maestra.

Los eventos pueden asociar a un objeto con otro. Veamos un ejemplo: En muchas organizaciones cuando un objeto es clasificado como un Empleado, este debe ser asociado con un Departamento.

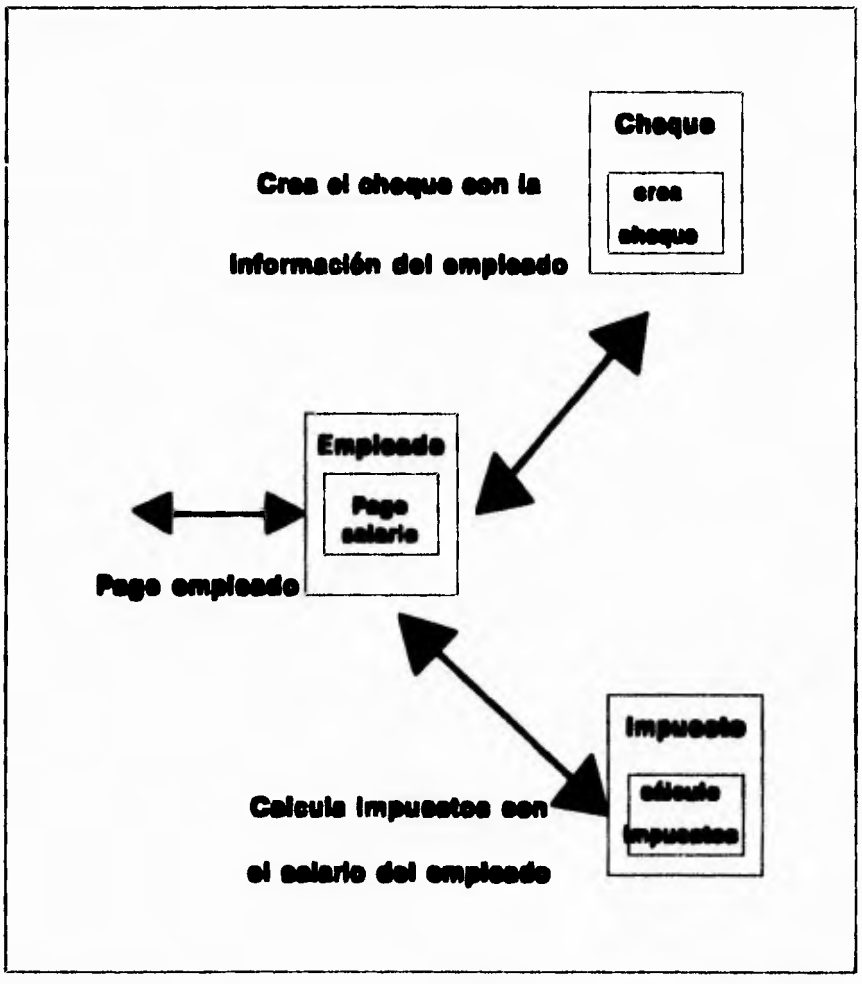
Algunos eventos requieren que otros eventos ocurran primero. Tal como:

Si un departamento debe ser cerrado, todos sus empleados deben de ser reubicados en otro departamento, o liquidados.

### 3.4.3 **Interacciones entre tipos de objetos.**

Los diagramas de transición son buenos para expresar el Ciclo de Vida de un objeto en particular. Sin embargo, muchos procesos involucran la interacción de varios tipos de objetos. Por ejemplo: Supongamos que existe un requerimiento de pago a un **Empleado**. Cuando esto ocurre hay que enviar una solicitud para calcular y regresar una cantidad de **impuesto**. Cuando este cambio de estado ha ocurrido, se deberá crear un objeto **Cheque**.

Diagrama de mensajes que se transfieren entre objetos:

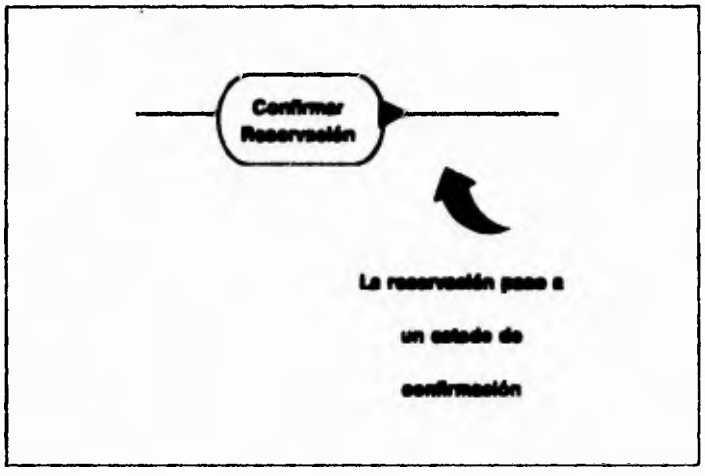



El resultado es que el estado del objeto Empleado cambia al de Empleado Pagado.

Cuando los diagramas son expresados de esta forma, es fácil ver cómo van a ser implementados en algún lenguaje orientado a objetos: los tipos de objetos se implementaran como clases; las operaciones llegarán a ser métodos.

Si el analista (con ayuda del usuario) es capaz de expresar las interacciones entre objetos de esta forma, el trabajo del diseñador se facilitará.

Las operaciones son representadas por rectángulos redondeados, y los eventos por triángulos negros conectados a la caja:

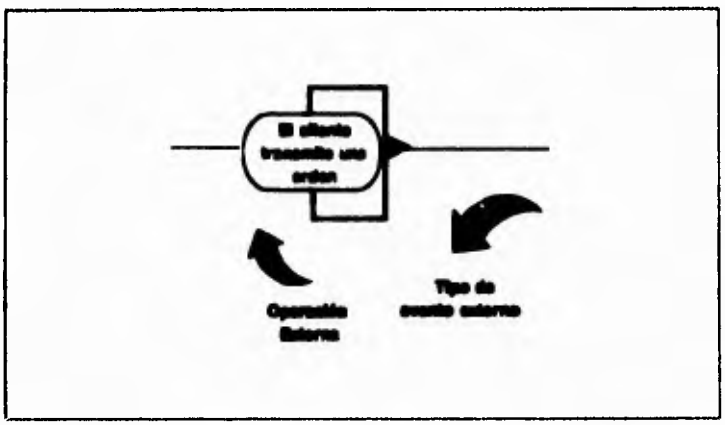



Los métodos en una clase manipulan solo los objetos de la clase, ellos no pueden accesar directamente los datos de un objeto en una clase diferente. Para usar los datos de una clase diferente, ellos deberán enviar una petición al objeto.

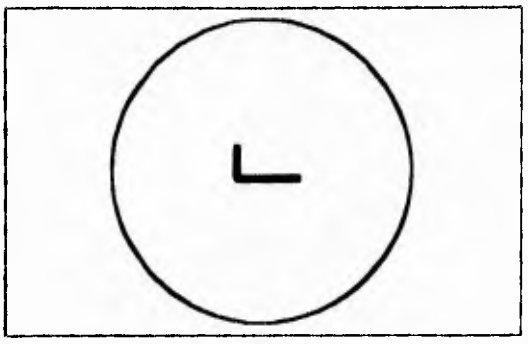
### 3.4.4 Fuentes externas de eventos.

Los eventos son cambios de estados que un sistema debe conocer y saber como reaccionar ante ellos. Muchas de las operaciones que producen estos eventos suelen ser externas.

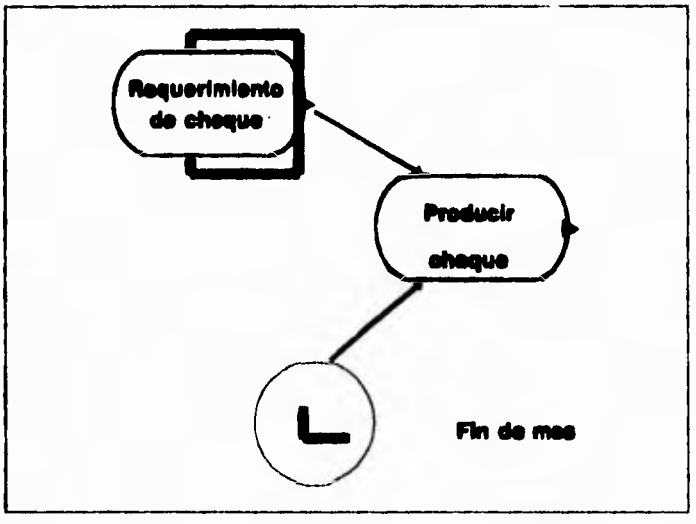
En este caso el símbolo de operación es dibujado como:



Un tipo especial de fuente externa es el "reloj externo", el cual indica que un proceso externo puede provocar que se ejecute una operación en un momento determinado, tal como: fin de quincena, principio de año, 25 de diciembre, etc. Es representado como:



Ejemplo:

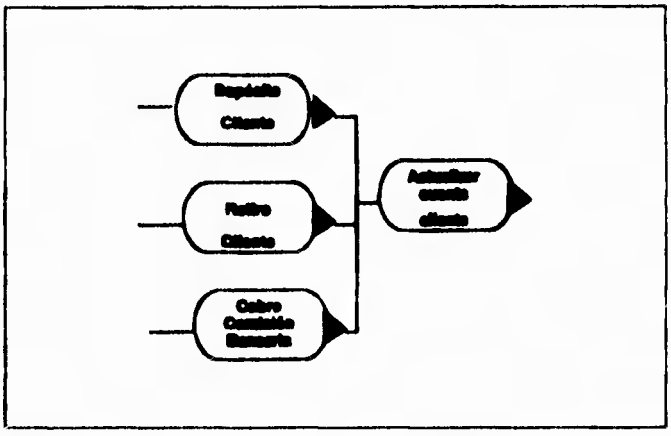



Un cheque será generado cada vez que sea fin de mes o cada que haya un requerimiento.

### 3.4.5 Reglas de disparo.

Cuando un evento ocurre, el estado de un objeto debe ser modificado, para ello es necesario que se realice una o más operaciones.

Las reglas de disparo definen las condicionantes para iniciar una operación cuando ocurre un evento.

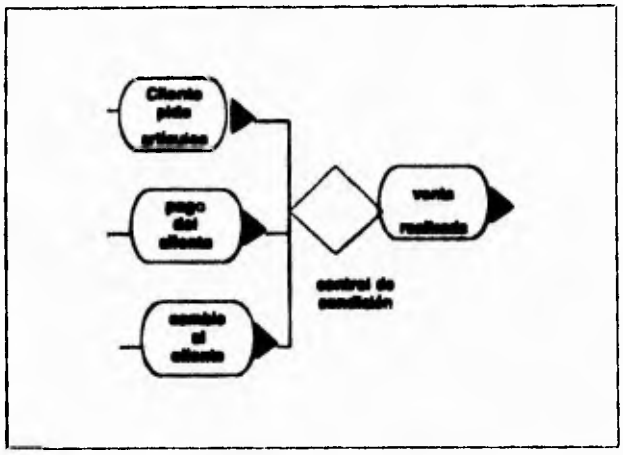


En la figura de arriba, se observa que la operación será "disparada" cada vez que ocurra alguno de los eventos indicados.

Sin embargo si fuera necesario una combinación de condiciones, se deberá poner un control de condición.

Cada vez que se llega a un control de condición se verificará si una condición dada es verdadera o falsa, si es verdadera el evento ocurrirá.

Ejemplo:

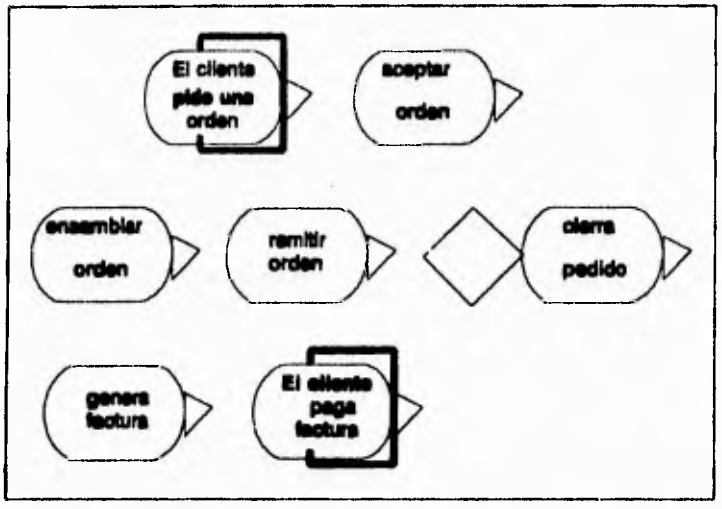


Para que una venta sea realizada, el cliente deberá solicitar el artículo que desea llevar, lo tendrá que pagar y la cajera deberá devolver al cliente su cambio (si es que lo hubiera).

**Esquemas de Eventos.**

Describen procesos en términos de eventos, disparadores, condiciones y operaciones.

Ejemplo:




**Diagramas de flujo de objetos.**

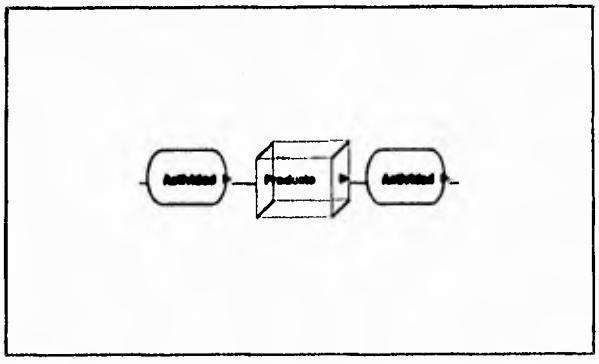
Existen ciertos procesos largos y complejos, los cuales si representaramos mediante esquemas de eventos no resultaría lo más conveniente.

Además por el momento, quizás solo se necesite un diagrama de alto nivel para entender la problemática. Esto es necesario sobre todo en la planeación a nivel estratégico.

En situaciones como estas, un Diagrama de flujo de objetos (DFO) puede ser de gran ayuda. Ellos nos indican los objetos, sus actividades y los intercambios que tienen con otros objetos.

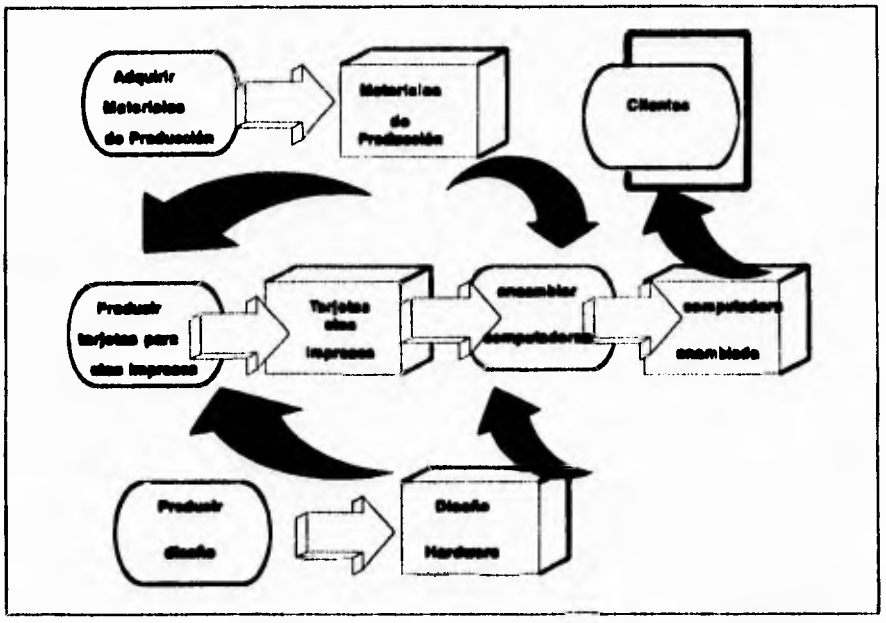
Mientras que en un Diagrama de flujo de datos (en la Orientación a Datos) las interfases se intercambian datos, los DFO representan cualquier cosa que se pasa de una actividad a otra, ya sean pedidos, partes, diseño, software, servicios o otros.

Se representan con las siguientes figuras:



El producto es el resultado final que satisface el propósito de la actividad. Sin embargo, los productos no solo se producen en áreas de la producción sino también se producen para el consumo por parte de otras actividades, que le añaden valor al producto consumido(para producir un nuevo producto mas complejo).

Ejemplo:




Los DFO describen objetos y la forma en la cual son producidos y comsumidos. Son útiles para planear la información estratégica del negocio. No obstante, si se requiere un mayor nivel de detalle sería apropiado un esquema de eventos.

**3.5 Diseño de la Estructura y Comportamiento del Objeto.**

Los lenguajes orientados a objetos, tienen estructuras de datos y métodos, juntos, pueden ser heredados y combinados dentro de unidades llamadas CLASES. Esta es la razón por la cual se describen estos dos puntos conjuntamente.

Los componentes que deben ser identificados en esta fase son:

- Las clases a implementar. Los tipos de objetos obtenidos en el Análisis de la Estructura del Objeto nos guiarán en esta decisión.

- La estructura que cada clase va a emplear.

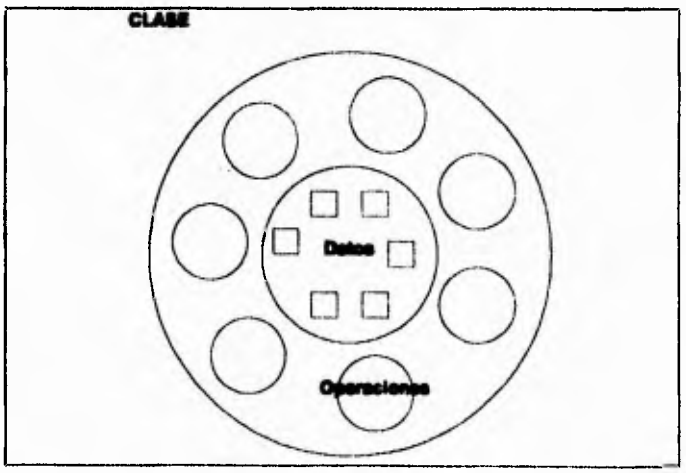
- Operaciones que cada clase presenta y los métodos asociados a ésta.

- Clases provenientes de otras, identificar la forma en la que van a ser implementadas, reconocer de que forma va a afectar esto a sus operaciones y métodos.

- Variantes existentes en las clases. (Lo mismo que, pero..).

Una clase es una implementación de un tipo de objeto. La clase especifica la estructura de los datos y las operaciones que pueden aplicarse a cada objeto.

Graficamente:

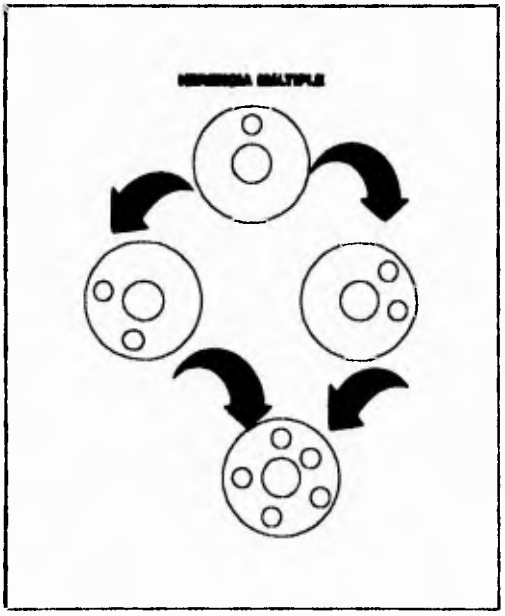


### 3.5.1 Herencia.

En el análisis ya se identificó si estos tipos de objetos están organizados en supertipos o subtipos. Ahora, en el diseño esta jerarquía es implementada usando la herencia.

La herencia hace posible que una clase comparta la estructura de datos y las operaciones de otras clases. La herencia es simple cuando una clase hereda los datos y la estructura de una clase base. La herencia múltiple es cuando una clase hereda los datos y estructura de más de una clase base.

Graficamente:

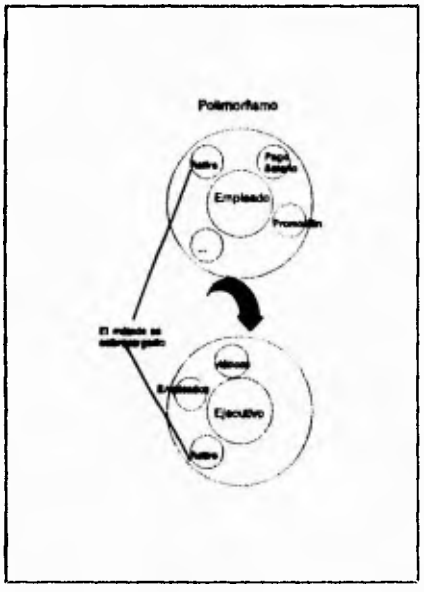


### 3.5.2 Polimorfismo.

Una de las principales ventajas de las técnicas orientadas a objetos es la reutilización de código.

Sin embargo, algunas operaciones pueden requerir adaptarse a ciertas necesidades particulares.

Ejemplo:




En la figura de arriba, la clase "Empleado" define una operación de retiro. En la implantación, esta operación es automáticamente heredada por todas las subclases de "Empleado". No obstante, una organización puede tener distintos métodos para retirar a un empleado o a un ejecutivo.

En esta situación, el método "retiro" del "Ejecutivo" redefine (sobrecarga) al método "retiro" del "Empleado", ya que se tratará de una forma u otra dependiendo el puesto que desempeñe la persona en la empresa.

Aunque estos métodos difieren, ellos realizan un mismo propósito. a esto es lo que se le conoce con el nombre de polimorfismo. Ya que una operación puede tomar diferentes formas de implantación dependiendo del tipo de objeto.

### 3.5.3 Lo mismo que, excepto...

En la práctica gran parte de la reusabilidad requiere que se modifiquen componentes reusables.

Un Arquitecto toma el diseño de un baño estándar y lo modifica para incluir una tina y un piso de mármol.

Las técnicas orientadas a objetos, deben permitir personalizar las clases. Las aplicaciones personalizadas deben ser rápidas, sin tener que entrometerse en el código de la clase.

Los diseñadores deben de anticipar aquellos aspectos que el usario pueda modificar, proporcionando herramientas para realizarlo.

El diseñador que crea una clase debe responder a la pregunta: Cómo es que esta clase pueda ser usada en el futuro?. Se deberá crear la clase de una forma tal, que pueda ser adaptada para futuras necesidades.

En un ambiente orientado a objetos adecuado, la mayor parte del código deberá ser construido de clases existentes o bien con la creación de nuevas clases para su futuro reuso. Todo relaciona la reusabilidad desde el pasado o la reusabilidad en el futuro.

**3.6 Conclusiones del capítulo.**

En el mundo de la Orientación a Objetos, Analistas, Diseñadores, Programadores y Usuarios piensan sobre un mismo modelo de la realidad basado en objetos y su comportamiento, el cambio entre el Análisis y el Diseño es más natural ya que se genera uno como consecuencia del otro, por ello resulta difícil señalar donde termina el análisis y comienza el diseño.

Sobresalen los métodos de Análisis y Diseño de J. Martin y J. Odell, Booch, Coad y Yourdon.

Aunque difieren un poco en los detalles, los métodos de diseño y análisis orientado a objetos comparten entre sí cuatro puntos básicos:

- 1. Identificar y describir a los objetos, sus atributos y relaciones.

- 2. Describir el comportamiento del objeto en respuesta a algún evento.

- 3. Describir algoritmos para implementar cada acción identificada en el paso anterior.

- 4. Validar el sistema completamente. Se deberá verificar la liga entre los procesos y los datos del objeto, para que éste funcione adecuadamente.

# CAPITULO 4. Programación en Windows.

## 4.1 Introducción.

El entorno de un programa o sistema operativo es el elemento de conexión entre la computadora y el usuario. Para decirlo de otra forma, se trata de aquello que aparece en la pantalla y el modo en que aparece.

El manejo del sistema operativo ha sido siempre una cuestión bastante compleja. Si el sistema operativo está correctamente instalado, aparece en la pantalla un discreto mensaje que indica la disposición del sistema operativo a recibir comandos u órdenes (C:\> simboliza el entorno de DOS). Sólo el conocimiento de los correspondientes comandos permite al usuario dar algunos pasos en el mundo de la computación. Pero no basta la auténtica avalancha de comandos del sistema operativo, por regla general, para obtener el resultado deseado, hace falta añadir a los comandos incontables parámetros. Por ello, el provecho que se obtenga de la computadora depende, en una gran medida, del conocimiento de los comandos y sus correspondientes parámetros.

El uso de programas creados para un determinado fin (edición de texto o datos, confección de gráficos), representó un alivio para los principiantes. Los comandos disponibles estaban, en su totalidad, representados en la pantalla, o bien ordenados en distintos niveles de menus más o menos profundos.

El sucesivo desarrollo del sistema operativo DOS, que condujo a las versiones 4 y 5, dió paso a la introducción del llamado Shell del DOS, un entorno de usuario para el sistema

operativo. Este entorno permitía, por primera vez, llevar a cabo operaciones a nivel del sistema operativo sin necesidad de consultar el manual. Las funciones más importantes están constantemente presentes en la pantalla, mientras el Shell del Dos está activo.

Este tipo de entorno de usuario, donde los comandos del programa aparecen en la pantalla representados meramente mediante texto, se denomina entorno basado en caracteres. Otros entornos basados en caracteres son los que ofrecen, por ejemplo, los programas Word 5.5, Norton, dBaseIV, etc.

Mucho antes de que saliera al mercado el primer Shell del DOS, la empresa Apple presentó una familia de computadoras cuyo sistema operativo estaba provisto de un entorno gráfico. En este caso el entorno gráfico significaba el uso de iconos y otros elementos gráficos para la representación de comandos y funciones. La representación gráfica permitía al principiante un acceso mucho más fácil a la informática, y al usuario iniciado un manejo intuitivo de cualquier programa, aunque fuera desconocido.

El desarrollo de estos entornos provocó un nuevo problema: Mientras numerosos programas competían por ofrecer el entorno más perfeccionado y fácil, el usuario se veía obligado a moverse a tientas por entre esa jungla de entornos, a menudo muy diversos.

Por ejemplo, existen entornos de programa que ofrecen el uso del ratón; otros, en cambio, se presentan provistos de una cantidad casi inimaginable de niveles de comandos (como Lotus). En muchos casos, había que avanzar a lo largo de hasta siete niveles para poder acceder a un determinado comando. Las diferentes maneras de seleccionar comandos dentro de diferentes

programas puede llegar a sumir en la confusión incluso a usuarios conocedores. La activación de comandos desde el teclado con ayuda de combinaciones de teclas, o la selección de un comando mediante las teclas del cursor representa un modo de manejo indudablemente claro. Pero si se trabaja con programas de muchos fabricantes diferentes, hay que adaptarse cada vez al modo de manejo de cada programa.

Poco a poco, cada fabricante ha ido desarrollando, para su gama de productos, un determinado concepto de manejo, con el resultado de que el manejo de sus productos se ajusta siempre al mismo esquema. Determinados comandos conducen, en los distintos productos, a las mismas operaciones. Hay determinadas teclas que tienen la misma función en todos los programas. El tipo de estructura de la pantalla se ajusta siempre al mismo principio.

En fin, en los desarrollos de los entornos del programa se han vertido una serie de consideraciones ergonómicas, didácticas y en gran parte, prácticas.

**4.2 Windows.**

En la actualidad el sistema operativo DOS está luchando con sistemas operativos más modernos y potentes, tales como el UNIX y el OS/2, para conservar el lugar preeminente que ocupa en el mundo de las computadoras personales.

Sus ventajas son muchas:

El conocimiento que de él tienen ya todos los usuarios de computadoras personales, su sencillez de manejo y, sobre todo, la gran cantidad de programas y utilidades existentes en el mercado. También tiene desventajas:

- Limitada potencia (especialmente en lo que se refiere a la memoria interna), la falta de un entorno gráfico intuitivo y la imposibilidad de utilizar multitarea.

- Precisamente ahí es donde entra Windows, que está creado para eliminar estas desventajas. Windows proporciona un entorno gráfico que crea un sistema de trabajo más cómodo, sencillo e intuitivo. También soluciona en gran medida los problemas de memoria interna, ya que efectúa una gestión muy avanzada de la misma, permitiendo emplear tanto la memoria extendida como la memoria expandida. Por otro lado, Windows realiza una multitarea virtual que permite ejecutar al mismo tiempo más de un programa o una aplicación.

Con Windows se ha creado un estándar, una nueva norma de trabajo para el sistema operativo DOS, proporcionando prestaciones nuevas y muy avanzadas que no tienen nada que envidiar a las que poseen los nuevos sistemas operativos. De esta manera se logra afianzar de forma muy seria el sistema operativo que todo el mundo sabe usar, el DOS.

Windows se diseñó originalmente a principios de la década de los 80's antes de la aparición de C++. Aunque los diseñadores se dieron cuenta de lo importante que era considerar los diversos elementos de una interfaz como objetos, no tuvieron más remedio que conformarse con emplear el código C tradicional para manipular dichos elementos. La mayoría de los recursos de Windows que pueden utilizar sus aplicaciones se podrían considerar como objetos.

Windows actúa de enlace entre el sistema operativo y el usuario, además, establece una conexión entre el sistema operativo y las aplicaciones Windows. Creando un sistema de

trabajo más cómodo, sencillo e intuitivo, incorporando además, unas potencias y características nuevas. Mientras los programas de otros fabricantes necesitan determinados archivos para el control de la pantalla, el ratón y la impresora, Windows se encarga de ese control de una sola vez para todos los programas que funcionen en él. Cuando se trabaja con una aplicación de Windows, por ejemplo Excel, dicha aplicación aprovecha el sistema de control de pantalla de Windows. Excel no posee un sistema de control de pantalla propio. De este modo se garantiza un diseño unitario de todos los programas.

Las características principales de Windows son: Las ventanas, la multitarea, la gestión de memoria, librerías dinámicas, vías de comunicación para el intercambio de datos, Interfaz múltiple de documentos.

### 4.2.1 VENTANAS.

Son recuadros de tamaño variable que aparecen en la pantalla y que contienen algún tipo de aplicación, que se ejecuta desde la propia ventana. Se puede tener abiertas varias ventanas al mismo tiempo, y se puede saltar de una a otra para activar la aplicación con la que se vaya a trabajar en cada momento. Las ventanas pueden solaparse en la pantalla, de manera que la que quede en un plano superior será la que este activa, es decir es la que se está utilizando. La ventaja de utilizar ventanas es que no es necesario salir de una aplicación para utilizar otra, sino que basta con cambiar de ventana y se activará la aplicación de la recién seleccionada. Por esta razón, se puede decir que Windows dispone de multicarga, es decir, que puede tener cargadas en memoria varias aplicaciones aunque sólo esté utilizando una de ellas en cada momento, quedando las demás en un estado latente o de espera. Las aplicaciones en estado latente se muestran en

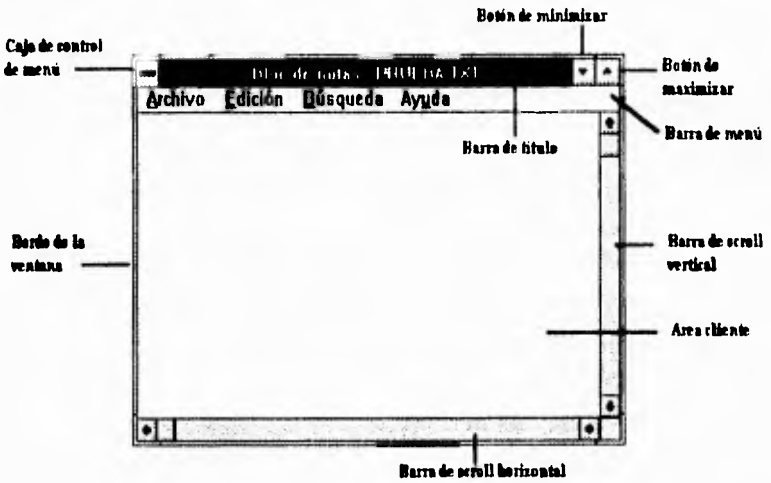
la pantalla mediante un icono que es un pequeño dibujo representativo de la aplicación.

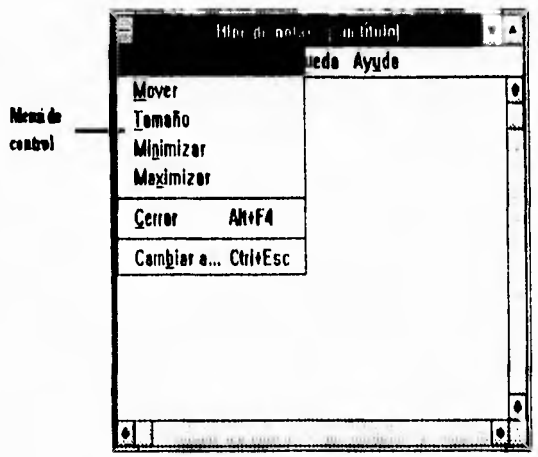
El número de ventanas que pueden estar abiertos al mismo tiempo depende de la potencia y capacidad de memoria del equipo que se tenga.

**Elementos de una Ventana.**

Una ventana es la parte más importante de la interfase de usuario. Los componentes de esta ventana son los siguientes :

- El borde de la ventana (border)
- El área cliente (client area)
- La caja de control de menú (system menu)
- El menú de control (control menu)
- Las barras de scroll (vertical y horizontal)
- Cajas de maximizar y minimizar (minimize/maximize-box)
- La barra de menú (menu bar)
- La barra de título (title bar)


Botón de minimizar
Caja de control de menú
Botón de maximizar
Archivo Edición Búsqueda Ayuda
Barra de título
Barra de menú
Borde de la ventana
Barra de scroll vertical
Area cliente
Barra de scroll horizontal


Menú de control
Mover
Tamaño
Minimizar
Maximizar
Cerrar Alt+F4
Cambiar a... Ctrl+Esc
Ayuda

### 4.2.2 MULTITAREA.

Es otra característica importante, que permite ejecutar más de una aplicación al mismo tiempo. Es decir, no sólo se pueden efectuar la carga simultánea de aplicaciones sino que, además, se pueden trabajar con más de una de ellas al mismo tiempo.

Sin embargo, la multitarea sólo es posible utilizarla en un Windows que trabaje con computadoras que tengan un procesador 80386 o superior, ya que no es una multitarea real, lo que hace Windows es crear una multitarea ficticia, a base de repartir el tiempo de procesador entre cada una de las aplicaciones. A esto se le llama multiplexaje en el tiempo. Es decir, cada aplicación simultánea utiliza el procesador durante una parte del tiempo, dejando el resto del tiempo a las demás, proceso que se repetirá continuamente. De esta manera se da la sensación de que realmente se están ejecutando las tareas al mismo tiempo.

Esta multiplexación en el tiempo hace que los procesos se ejecuten más lentamente cuantas más aplicaciones esten funcionando a la vez, puesto que el tiempo deberá repartirse entre mas elementos.

### 4.2.3 GESTION DE MEMORIA.

Windows puede manejar memoria interna por encima de los 640 Kbytes que tiene como limite el DOS. Además de la memoria convencional, los tipos de memoria que maneja Windows son la memoria extendida, la expandida y virtual.

ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

A cada dirección de memoria de la computadora personal se accede a través de un bus de direcciones. El procesador 8088 se comunica con las direcciones de memoria a través de 20 canales. Con ello, el procesador es capaz de gestionar 1,048,576 direcciones de memoria, es decir, 1 MByte de memoria. Asi pues, un procesador XT sólo puede direccionar un máximo de 1 MByte.

Pero el área de memoria directamente direccionable sufre una nueva división debido al hecho de que se carga el sistema operativo DOS. Trabajando con la arquitectura de una PC, el sistema operativo DOS puede poner a disposición de las aplicaciones un máximo de 640 KBytes. A esto se le conoce con el nombre de memoria convencional, los restantes 384 KBytes por encima de esta frontera están reservados para la memoria del video, el BIOS y otras extensiones como por ejemplo ROM adicional. Pero estos elementos no ocupan por completo el espacio disponible en esta área. Existe un espacio de 64 KBytes situados entre los 768 y 832 KBytes que puede ser aprovechado. A esto se le llama memoria expandida, la cual se compone de cuatro bloques de memoria de 16KBytes llamados páginas, sólo puede ser utilizada valiendose de un programa, concretamente un driver. El estándar LIM/EMS (EMS=Expanded Memory Specification) desarrollado por Lotus, Intel y Microsoft, representa el fundamento para el uso de la memoria expandida.

Los procesadores 80286, 80386, 80486 son capaces de direccionar más de 1 MByte de Memoria. Esta memoria situada a partir de 1 MByte se le llama memoria extendida, la cual es muy rápida. Sin embargo, necesita un controlador especial (llamado extended memory manager) que gestiona dicha zona de la memoria. Windows dispone de un controlador de este tipo: HIMEM.SYS.

Los programas escritos para Windows ahora, incluso pueden acceder a una determinada zona del disco duro, y emplearla como memoria. Para el acceso correcto a esta memoria virtual se ha incluido en Windows una gestión de memoria especial denominado Manejador de Memoria Virtual (VMM). Posee la tarea de dividir la memoria principal completa internamente en trozos de 4KBytes cada uno, y descargar al disco en caso necesario aquellas páginas que llevan más tiempo sin utilizarce. El VMM gestiona la llamada tabla de páginas, que contiene referencias a todas las páginas que se encuentran tanto en memoria como en disco duro. Además esta tabla contiene banderas para cada página, que indican si después de cargar la página se accedió a ésta (atributo accesed) y si se trató de un acceso de escritura (atributo dirty). Si se debe cargar una nueva página del disco duro en memoria, pero no hay espacio disponible, Windows combrueba que estos dos atributos no estén activos, únicamente en este caso es cuando se puede colocar la página correspondiente en el disco.

### 4.2.4 LIBRERIAS DINAMICAS.

Las definiciones de todas las funciones Windows se encuentran en librerías. Estas librerías no están incluídas de forma fija en el programa como las librerías estáticas. Sólo al ejecutar el programa, o en caso necesario son cargadas estas librerías.

Una DLL (Dynamic Link Library) es un módulo que contiene funciones y recursos, que pueden ser utilizados por una aplicación Windows.

Windows se compone de tres DLL's:

- **KERNEL.EXE.** Es responsable del manejo de la memoria, recursos y multitarea.

- **USER.EXE.** Controla el manejo de Windows, gestiona todas las entradas.

- **GDI.EXE.** Suministra el entorno gráfico y se encarga de todas las salidas.

A ello se añaden varios controladores de dispositivos que también han sido realizados como DLL. Por ejemplo: DISPLAY.DRV para el controlador de la pantalla, MOUSE.DRV para el manejo del ratón. El ligado dinámico ocurre sólo cuando se llama una función de la DLL. En el código de la aplicación que emplea las funciones DLL sólo se colocan referencias cruzadas de la librería especificada. Varias aplicaciones pueden acceder al mismo objeto en una DLL. Este objeto puede ser una función o un recurso. La DLL ha de existir una sola vez en la memoria. Con ello se ahorra espacio y las funciones que están definidas en la DLL se pueden modificar con el mínimo esfuerzo, ya que no se han de recompilar y ligar con cada programa, sino sólo la DLL.

Una DLL no posee una dirección de entrada global y reacciona a las solicitudes de la aplicación que desea hacer uso de ella. De modo que cada función allí definida se direcciona individualmente y espera ser llamada por la aplicación.

## 4.3 VIAS DE COMUNICACION PARA EL INTERCAMBIO DE DATOS.

Windows ha crecido tanto, que cada vez se pueden realizar proyectos más grandes. Por ello cada día se encuentran una nueva gama de productos desarrollados por diferentes personas. Si estos programas deben comunicarse unos con otros, primero sería necesario establecer vías de comunicación fijas en los programas. Microsoft ha logrado establecer un estándar para el intercambio de datos bajo Windows con DDE y OLE.

### 4.3.1 DDE (Dynamic Data Exchange).

Para poder intercambiar datos entre programas es necesario establecer una comunicación entre ellos. A causa de ello nació el concepto DDE. No significa otra cosa que un intercambio de datos dinámico (Dynamic Data Exchange). El término "dinámico" indica que una comunicación de los programas puede comenzar y finalizar en cualquier momento. DDE es un protocolo estándar de comunicación que se basa en el intercambio de mensajes. Para la programación existe una colección de funciones DDE API, que permiten la comunicación de programa a programa. Ya que DDE es un protocolo homogéneo, cualquiera puede participar en una comunicación, si respeta las reglas. Las aplicaciones más dispares y diferentes pueden comunicarse entre sí, especialmente para los casos en que se realiza una comunicación entre programas de forma automática, es decir, de forma completamente independiente del usuario, DDE es útil.

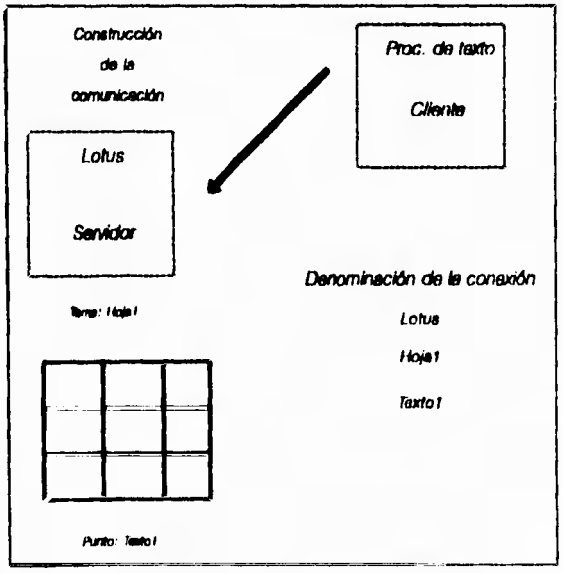
La aplicación que comienza una comunicación es llamada Cliente. El Cliente es aquel que quiere obtener una prestación. Este servicio lo pide al Servidor. Un programa puede tener más de una conexión en un momento dado y además puede ser cliente y servidor a la vez. En este caso la aplicación ha de obtener

prestaciones (Cliente) pero simultáneamente ha de realizar servicios para otros (Servidor).

**Creación de una conexión de datos.**

Para que dos aplicaciones puedan intercambiar datos, primero se ha de establecer una comunicación DDE. La construcción de la conexión ha de ser iniciada obligatoriamente por el Cliente. A causa de la petición, contestará a continuación el Servidor. Este pondrá los datos pedidos a disposición, bajo petición. El Cliente deberá especificar el nombre del servidor (aplicación con la que se quiere comunicar), el tema y punto de conversación.

Cuando el servidor obtiene una petición de inicio de conversación DDE, se comprueba el tema y el punto para ver si son conocidos. Una vez que se construyó la conexión, ya no es permitido cambiar el servidor ni el tema. La combinación del nombre de servidor y tema identifican la conexión de manera unívoca y permanece hasta el final de la comunicación. Si se llegara a modificar alguno, la comunicación se interrumpe de inmediato.

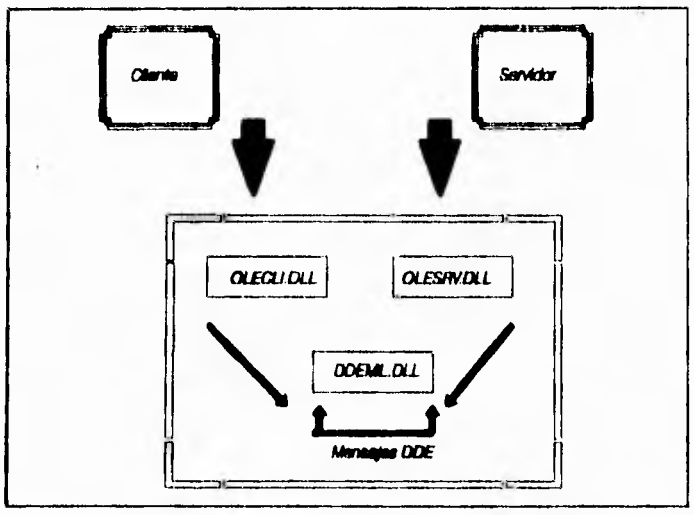


### 4.3.2 OLE (Object Linking and Embedding).

El camino de intercambiar datos entre programa gráfico y programa de texto, ya funcionaba bien a través del portapapeles. Con él, se podía insertar un gráfico dentro de un texto, y como resultado se generaba un texto con un dibujo copiado en él. Si el gráfico se modificaba era necesario tomar el programa gráfico (como Paintbrush) y realizar las correcciones, a continuación se ha de hacer otra vez el camino a través del protapapeles. Cuántos más errores existen, tanto más frecuente es la secuencia manual. El proceso era muy poco práctico, estaría bien que el dibujo se copiara mediante el portapapeles y que DDE se encargará automáticamente de la actualización de los datos. Y esto es exactamente lo que hace OLE.

Al igual que DDE también OLE es un protocolo estándar para intercambiar datos y comandos. La realización de estos sistemas ocurre puramente a través del intercambio de mensajes Windows. Por ello no se necesitan soporte mediante las llamadas al sistema operativo. OLE está realizado de forma, que se construye sobre el protocolo DDE. Esto se realiza mediante el paso de cadenas de comandos mediante DDE. El programador o usuario no han de preocuparse de esta secuencia.

La programación de una aplicación OLE y DDE se realiza con la ayuda de las funciones Windows API. Estas funciones se procesan mediante DLL fuera de Windows. Para OLE esta es la librería OLECLI.DLL para el Cliente y la librería OLESVR.DLL para el Servidor. Todas las llamadas de función DDE se ejecutan mediante DDEML.DLL.




Como DDE es un subconjunto de OLE, éste puede hacer todo lo que hace DDE. Un programa que soporta OLE, puede por ello construir una comunicación con ayuda del tema, además los datos se obtienen mediante especificación del punto por el cliente.

También es posible un envío de datos (mediante el Servidor) al Cliente, igual que una obtención de datos por el Cliente del Servidor. Para que puedan pasar instrucciones al Servidor, es posible, al igual que en DDE, que el Cliente envie una línea de comando como cadena de caracteres al Servidor. También OLE (a diferencia de DDE) efectúa de manera automática las posibles traducciones de los formatos habituales como mapa de bits, mapa de bits independientes de dispositivos y archivos.

DDE sigue empleándose para aplicaciones que necesitan muchas conexiones simultáneamente, especialmente para muchos puntos, si estos se actualizan frecuentemente. La realización de un sistema

de información en tiempo real es un caso típico, para que el DDE tenga su derecho a existir.

### 4.3.3 Conceptos OLE importantes.

- **Objeto OLE.**

La forma de pensar de OLE se apoya mucho en la orientación al objeto. La tarea de OLE ya no consiste en intercambiar datos, sino objetos completos. Estos objetos son datos encapsulados de los tipos mas variados, que se pueden componer por ejemplo de texto, imágenes, sonidos. El encapsulado de los datos en objetos OLE, lleva a que estos se puedan mantener y utilizar por cualquier aplicación OLE participante, sin tener que conocer su estructura exacta.

- **Documento Contenedor.**

Para guardar los objetos OLE más diversos se emplea el documento contenedor. En este se pueden recolectar objetos y rellenar mediante informaciones propias.

- **Cliente.**

Programa que esta en situación de tomar objetos OLE, mostrarlos y guardarlos.

- **Servidor.**

Los objetos a su vez se generan mediante una aplicación Servidor, que pone a disposición datos, y que permite su modificación. El Servidor es el único que ha de conocer la

estructura exacta de los datos en el objeto.

- **Incrustar.(Embedding)**

Si un objeto se ha de convertir en una parte fija de un documento, se le llama incrustación. El objeto se encuentra completamente en el documento contenedor y ligado separadamente a cada una de las aplicaciones. Ejemplo: Si un objeto de Excel esta incorporado a Paintbrush y Write, podemos modificarlos separadamente en Write, sin que esta modificación tenga efecto automáticamente en Paintbrush.

- **Vincular.(Linking)**

Un objeto sólo se encuentra conectado a los datos originales que hay en al archivo mediante referencias. A pesar de ello, el objeto puede ser mostrado por un cliente OLE. El archivo se puede actualizar por el Cliente OLE dependiendo de la opción de conexión de forma automática. Ejemplo: Si un objeto de Excel se conecta con Paintbrush y Write, cualquier modificación del objeto se reflejara tanto en la aplicación original (Excel) como en Paintbrush y Write.

- **Palabra de acción.(Verbs).**

Para que un cliente OLE pueda editar un objeto incrustado (pegado) o un objeto vinculado, sin conocer la estructura de los datos, se necesita una palabra de acción (Verb). Estas palabras de acción le indican al Servidor el deseo de edición.

▸ **Trabajo conjunto entre Cliente y Servidor.**

La aplicación principal del OLE consiste en pegar objetos de forma permanente y vincularlos. Esto ocurre reconstruyendo la conexión entre el Cliente y el Servidor cada vez que se activa un objeto OLE. Esta propiedad no la podía cumplir DDE, ya que una conexión permanente se debía realizar por el usuario y una conexión no permanente por la aplicación.

## 4.4 INTERFAZ DE DOCUMENTOS MULTIPLE. (MDI).

Es una interfase estándar para aplicaciones Windows la cual permite al usuario trabajar simultáneamente con diferentes documentos. Cada documento aparece en una ventana propia. Ejemplos de MDI son el Program Manager y el File Manager de Windows.

En anteriores versiones de Windows (antes de la 3.0), la creación de aplicaciones MDI era muy laboriosa, ya que el programador se había de encargar de todos los detalles. Windows 3.X quita todo ese trabajo de gestión de ventanas MDI. Pero la inclusión de las funciones MDI significa a pesar de todo, un esfuerzo adicional con respecto a las aplicaciones normales.

Una aplicación MDI se compone de tres tipos de ventanas:

**1. Marco. (Frame)**. Es la ventana principal de una aplicación MDI y posee casi siempre un menú. Se encarga de la construcción y manejo de la ventana Cliente y de las ventanas Hijo, así como de procesar los items del menú.

**2. Cliente. (Client).** Es el "fondo" de la ventana Marco, sobre la cual aparecerán las ventanas Hijo. Su función es controlar las ventanas Hijo.

**3. Hijo. (Child).** Se parecen mucho a una ventana principal normal, ya que pueden poseer como ella botón para minimizar y maximizar, pueden ser modificadas de tamaño, etc. Sólo que no se pueden sacar de la ventana Marco. No pueden poseer un menú propio, así que sus funciones son controladas por el menú de la ventana Marco. Las ventanas hijo se controlan en una aplicación MDI mediante mensajes especiales. Frecuentemente los mensajes sólo tienen efecto sobre la ventana activa.

### 4.5 MENSAJES EN WINDOWS.

Windows es un sistema orientado a mensajes, es decir, todas las informaciones dentro del sistema se intercambian mediante el envío y la recepción de mensajes.

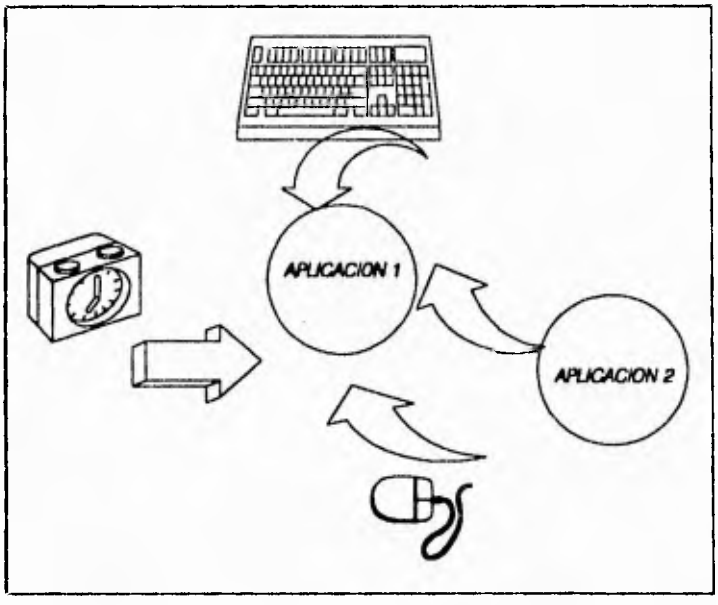
Todos los mensajes son gestionados por Windows con ayuda de Colas de Mensajes. Para todo el sistema existe una cola de mensajes del sistema y para cada aplicación una cola de mensajes de la aplicación.

Cola de mensajes del sistema: En ella se colocan todos los eventos de ratón, teclado y temporizador con ayuda de la librería USER.EXE y controladores. Otra parte de la librería USER.EXE obtiene de nuevo los mensajes de la cola, los traduce en una estructura predefinida MSG y los deposita en la cola de mensajes de la aplicación.

La estructura MSG se compone de 6 campos:

1. HWindow. Identifica la ventana que ha de recibir el mensaje.
2. Message. Indica de qué mensaje se trata.
3,4. WParam, LParam. Contienen información adicional del mensaje, su contenido varía de acuerdo al mensaje.
5. Time. Indica el momento en que se pasó el mensaje.
6. Pt. Indica en que posición se encontraba el ratón, cuando se pasó el mensaje.

Mensajes en Windows:



## 4.6 APLICACIONES BAJO WINDOWS.

Las principales características de un programa que puede ejecutarse bajo ambiente Windows son:

- Debe estar en un formato .EXE.
- No puede ejecutarse bajo DOS.
- Puede correr simúltaneamente con otras aplicaciones sean o no aplicaciones de Windows, incluyendo otras instancias de ella misma.
- Puede comunicar y compartir datos con otras aplicaciones de Windows.

Windows ofrece muchos beneficios tanto a usuarios como desarrolladores.

Beneficios para los usuarios:

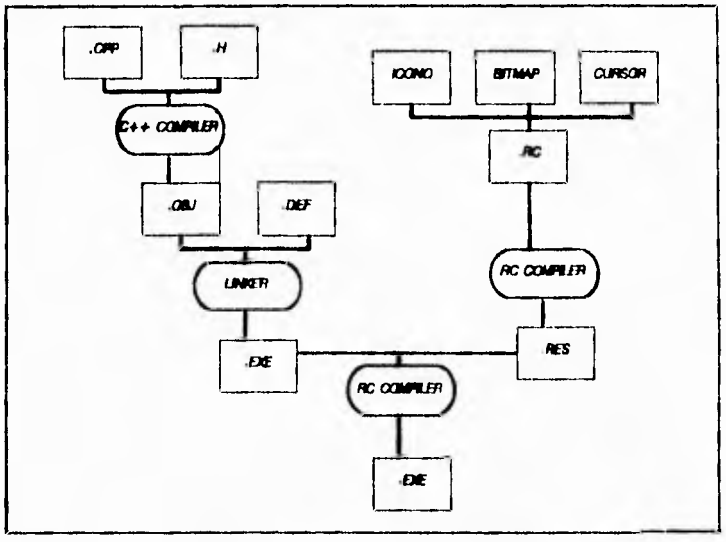
- Operación estandar y predecible. Al conocer el funcionamiento de una aplicación en Windows, se conoce automáticamente como funcionan las demas.
- Drivers para el soporte de periféricos.
- Comunicación entre procesos.
- Multitarea.
- Acceso a más memoria.

Beneficios para los desarrolladores:

- Las aplicaciones gráficas corren en todos los adaptadores estándar gráficos.
- Soporte inmediato para un gran rango de impresoras, monitores, mouse, etc.

- Una rica librería de rutinas gráficas.
- Manejo de más memoria para programas grandes.
- Soporte para menús, iconos, bitmaps y más.

## 4.7 COMPONENTES DE UN PROGRAMA WINDOWS.




### 4.7.1 Module Definition File (DEF).

Además del texto fuente se ha de crear un archivo de definición, que necesita el ligador, para poder enlazar el texto fuente compilado con Windows y otras librerías para que se pueda generar una aplicación Windows ejecutable.

Campos que tiene este archivo:

| Concepto | Descripción |
|---|---|
| NAME/LIBRARY | Nombre del módulo o DLL |
| DESCRIPTION | Comentario |
| DATA | Atributos del segmento de datos |
| CODE | Atributos del segmento de código |
| HEAPSIZE | Tamaño del heap local |
| STACKSIZE | Tamaño del stack local |
| SEGMENT | Segmento de código adicional |
| EXPORTS | Funciones públicas para otros programas |
| IMPORTS | Funciones requeridas para su ejecución |
| EXETYPE | Windows u OS/2 |
| STUB | Programa a ejecutar sin Windows |

Descripción de estos campos:

- **NAME:** El nombre debe ser siempre idéntico con el nombre de la aplicación Windows, sino el ligador dará un mensaje de aviso.
- **DATA, CODE:** Si la aplicación momentáneante no está en la posesión del CPU, y Windows necesita espacio para otros programas, Windows puede desplazar estos dos segmentos en la memoria, siempre y cuando se les ponga la opción de MOVEABLE.
- **EXPORTS:** Define los nombres y opcionalmente los números ordinales de todas las funciones a exportar. Las funciones se han de exportar si son llamadas por Windows u otra aplicación. Todas las funciones que llama Windows, se denominan como funciones CALLBACK. El número ordinal es una identificación adicional, y puede adoptar cualquier valor entero.

▸ **IMPORTS:** Aquí se han de indicar todas las funciones que son llamadas por la aplicación, pero que no están definidas en su archivo fuente ni en las librerías enlazadas estáticamente a esta aplicación. Estas funciones están definidas en las Librerias de Ligado Dinámico (DLL).
▸ **STUB:** El programa WINSTUB.EXE ya existe, y se llama cuando se intenta ejecutar la aplicación Windows desde DOS. WINSTUB.EXE visulaiza en la pantalla la frase "Este programa requiere Microsoft Windows" y termina la aplicación inmediatamente.

### 4.7.2 Recursos.

Aparte de los segmentos de código y datos en una aplicación Windows también puede poseer segmentos de recursos, que también está incluido en el archivo .EXE. Los recursos son datos que habitualmente no se pueden modificar durante el tiempo de ejecución. Los segmentos de recursos, al igual que los segmentos de código sólo existen una vez, de modo que varias instancias de la misma aplicación los han de compartir.

Los recursos se describen en un archivo de texto ASCII con extensión .RC. Un compilador especial, el Resource Compiler (RC.EXE) traduce el archivo RC en un formato binario, lo añade al archivo .EXE y crea las referencias en la tabla de recursos, que se encuentra en la zona de cabecera del programa. La estructura de los recursos en el archivo .RC tiene el siguiente aspecto: Nombre Tipo Datos.

Existen 9 tipos distintos de recursos:

1. **Icono.** Es una pequeña imagen que representa a la aplicación si esta se minimiza.

2. **Cursor.** figura que normalmente representa los movimientos del ratón en la pantalla.

3. **Bitmaps.** Imágenes de pixeles que se usan para dibujar en la pantalla o para crear patrones y pinceles.

4. **Tabla para string.** Se emplea para guardar cadenas de caracteres independientes del segmento de datos. Son comúnmente utilizadas para traducir el programa a diferentes idiomas.

5. **Menús.** Se componen de una serie de opciones, que se muestran en una o varias líneas por de bajo de la barra de títulos de una ventana.

6. **Aceleradores.** Son teclas con las que se puede seleccionar rápidamente alguna opción del menú.

7. **Cajas de diálogo.** Es una ventana mediante la cual se puede intercambiar información con el usuario. Ejemplo: Saber que tipo de impresora tiene el usuario.

8. **Recurso definido por el usuario.** El usuario puede definir sus propios recursos.

9. **Cajas de mensajes.** Posee una línea de título y un texto estático. Podría servir por ejemplo para indicar algún error en la aplicación.

## 4.8 PROGRAMANDO BAJO AMBIENTE WINDOWS.

Nunca ha sido sencilla la programación en Windows, desarrollar una simple aplicación bajo ambiente Windows puede ser una tarea muy complicada. Es irónico que la mayoría de los programas Windows se hayan escrito en código C tradicional, no obstante que Windows se considere orientado a objetos. Al utilizar el lenguaje C se producen programas Windows típicos que tienen una enorme proposición switch en la función principal de llamada automática de la ventana. Lo anterior no sólo produce un programa ilegible, sino también de dificil mantenimiento y depuración. Hay otras áreas que ofrecen dificultades especiales a los diseñadores de aplicaciones, como son el uso de la memoria, las bibliotecas de enlace dinámico, programas de interfaz con múltiples documentos(MDI).

Borland vino en ayuda del diseñador de aplicaciones Windows proporcionando un marco de aplicaciones denominado Biblioteca ObjectWindows, o más brevemente, OWL. Con ayuda de ObjectWindows se facilita considerablemente el proceso de escribir los programas Windows, se evita que aparezcan diversos detalles de bajo nivel en el código de las aplicaciones, permite concentrar el trabajo en las funciones de la aplicación más que en la forma.

El marco de trabajo de ObjectWindows permite usar objetos para representar elementos complejos de un programa Windows. Con estos objetos se encapsulan datos que todas las ventanas requieren, permiten ejecutar operaciones comunes, se da una respuesta automática a los mensajes de Windows. Como resultado de esto, se pueden escribir programas Windows con menos tiempo y esfuerzo.

Específicamente ObjectWindows proporciona 3 facilidades:

- Encapsulado.

ObjectWindows proporciona objetos que definen datos y métodos para ventanas, cajas de diálogo y controles de Windows.

- Abstracción de funciones Windows.

Windows es un entorno gráfico muy poderoso, pero también muy complejo. La interacción entre las aplicaciones del usuario y Windows se efectúa mediante un grupo de más de 600 funciones a las cuales se les denomina API (Interfaz de programas de aplicación). Cada función puede tomar una gran cantidad de parámetros de distinto tipo.

Aunque estas funciones pueden ser llamadas directamente en C++, ObjectWindows simplifica la tarea, muchos de los parámetros que las funciones de Windows requieren ya han sido almacenados en los datos de los objetos por lo tanto las funciones de Windows pueden tomar estos datos agrupando la llamada de funciones relacionadas en una sola, la cual ejecuta tareas de alto nivel.

- Respuesta automática a mensajes. La forma tradicional de responder a los mensajes Windows consiste en comparar el mensaje generado con una lista predefinida de mensajes a los cuales la aplicación debe responder (switch). Ejemplo:

```
switch(mensaje)
{
    case WM_BUTTONDBLCLK:
       Aqui se definen las acciones a realizar cuando el usuario
      presiona un doble-click del botón izquierdo del mouse.
```

```
    case WM_CLOSE:
       Operaciones a ejecutar cuando el usuario quiere cerrar
       la ventana.
}
```

Obviamente cuando se quiere responder a 200 diferentes mensajes de Windows, el código del programa crece en alarmante proporción.

La aplicación puede recibir cientos de mensajes en tan sólo unos cuantos minutos, procesar y responder a los mensajes es una parte crítica para el buen funcionamiento del sistema.

Los objetos con la habilidad de heredar y redefinir el comportamiento, pueden ejecutar la labor de responder a un evento(mensaje de Windows). ObjectWindows toma el mensaje de Windows y lo pasa a una función miembro de C++. Por lo tanto, sólo se requiere definir una función para cada mensaje que la aplicación requiera responder, estas funciones son llamadas **funciones de respuesta a mensajes**. Para relacionar un mensaje de Windows con una función de respuesta a mensajes se utiliza una herramienta del compilador de Borland C++ llamada Tablas virtuales despachadoras dinámicas. Al declarar una función de respuesta a mensajes, se debe especificar un índice despachador.

Ejemplo:

```
virtual void Pinta() = [110];
```

Pinta es definida como una función de respuesta a mensajes, cuyo índice despachador es 110.

La aplicación ObjectWindows dirige automáticamente el mensaje de Windows a aquella función cuyo índice despachador sea igual al valor del mensaje, de esta forma se simplifica el código del programa.

### 4.8.1 Anatomía de una aplicación ObjectWindows.

Es necesario que todo programa escrito con ayuda de ObjectWindos soporte al menos dos clases de objetos: un objeto de aplicación y un objeto de ventana. La aplicación es un marco de trabajo complejo encargado de administrar todos los objetos del programa, entre otros: menús, ventanas y cajas de diálogo. Todo programa debe construirse con base en una clase de aplicación derivada de la clase TApplication. A continuación se presenta el código necesario para poder desplegar una ventana con el título "Hola Mundo":

```
#include <owl.h>

class TInicio : public TApplication
{
 public:
  TInicio(LPSTR AName, HANDLE hInstance, HANDLE hPrevInstance,
         LPSTR lpCmdLine, int nCmdShow) : TApplication(AName,
            hInstance, hPrevInstance, lpCmdLine, nCmdShow) { };
 virtual void InitMainWindow();
}

void TInicio::InitMainWindow()
{
  MainWindow = new TWindow(NULL,"Hola Mundo");
}
```

```
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR
                   lpmCmdLine, int nCmdShow)
{
  TInicio Aplica("Aplicacion1", hInstance, hPrevInstance,
                   lpCmdLine, nCmdShow);
  Aplica.Run():
  return Aplica.Status;
}
```

Analizando cada parte:

▸ Cada programa Objectindows debe incluir este header.

```
#include <owl.h>
```

▸ Se define una clase derivada de TApplication, class TInicio :

```
public TApplication
{
 public:
  TInicio(LPSTR AName, HANDLE hInstance, HANDLE hPrevInstance,
         LPSTR lpCmdLine, int nCmdShow) : TApplication(AName,
            hInstance, hPrevInstance, lpCmdLine, nCmdShow) { };
 virtual void InitMainWindow();
}
```

▸ Se construye el objeto Ventana Principal, el cual es almacenado en el dato miembro MainWindow del objeto aplicación.

```
void TInicio::InitMainWindow()
{
  MainWindow = new TWindow(NULL,"Hola Mundo");
}
```

• Ninguna aplicación Windows puede funcionar sin la función WinMain, ya que esta es el punto de entrada. Esta función tiene las siguientes tareas: Inicialización, entre otras, de las clases de ventanas, crear y visualizar una ventana.

WinMain tiene siempre la misma estructura:

```
int PASCAL WinMain (hInstance, hPrevInstance, lpszCmdLine,
                    nCmdShow);
```

Significado de estas palabras:

**PASCAL:** La convención PASCAL indica que los parámetros han de colocarse en el stack de derecha a izquierda, y que la función llamada limpia el stack.
**hInstance, hPrevInstance:** Estas variables permiten identificar mediante un número único el programa que acaba de ejecutarse bajo Windows. Si el programa se ejecuta una segunda vez, esta copia obtiene una nueva hInstance (instancia). Con ello cada copia es una instancia nueva. La variable hPrevInstance es un manejador a la instancia anterior. Si el programa es ejecutado por primera vez, hPrevInstance tiene el valor de 0. Estos parámetros son gestionados por Windows.
**lpCmdLine:** Es un puntero far, que apunta a una cadena que termina con NULL. Aquí se pueden pasar argumentos al iniciar el programa.
**nCmdShow:** Este valor entero determina si el programa se ha de ejecutar como ventana o como icono, dependiendo de la selección del usuario.

• Esta parte construye el objeto aplicación, se inicializan los datos miembros del objeto.

```
TInicio Aplica("Aplicacion1", hInstance, hPrevInstance,
                lpCmdLine, nCmdShow);
```

▸ Se llama a la función miembro Run() de la aplicación, la cual llamará a las funciones InitApplication() e InitInstance() para ejecutar las inicializacion de la primera instancia y de cada instancia. Después es llamada la función InitMainWindow() para crear la ventana principal.

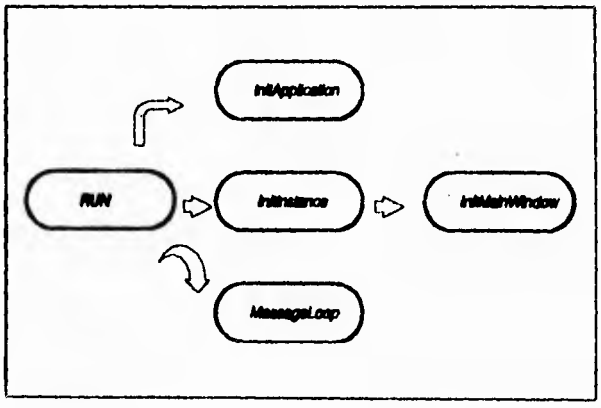
▸ Run() coloca la aplicación en movimiento por medio de la llamada a MessageLoop() para comenzar a procesar los mensajes de Windows.

```
Aplica.Run():
```

▸ Por último se regresa el status final de la aplicación. Esto es necesario porque WinMain debe regresar un entero. Este valor también es útil para verificar como terminó la aplicación, ya que un valor distinto de 0 indicará que hubo error.

```
return Aplica.Status;
}
```

Gráficamente:



La única funcion miembro que se requiere escribir es InitMainWindow(), también pueden redefinirse InitInstance() o InitApplication() para ejecutar alguna tarea en especial al momento de inicializar la primera o demás instancias.

Windows permite ejecutar más de una vez el mismo programa, a cada una de estas copias se les llama instancias. Se puede definir algún procedimiento para la primera vez que corre la aplicación redefiniendo a la función InitApplication(). Cabe señalar que si el usuario comienza y termina una aplicación, e inicia de nuevo, esta instancia es considera la primera. Asimismo, puede definirse una función al momento en que son creadas las instancias (incluyendo la primera) como cargar una tabla de aceleradores.

Las funciones InitApplication() e InitInstance() sólo deben ejecutar tareas referentes a la inicialización del objeto aplicación. La inicialización del objeto Ventana Principal debe ser hecha en su constructor y en su función miembro SetupWindow.

Por medio de la función MessageLoop se procesan todos los mensajes provenientes de Windows. Antes de que la aplicación termine es necesario terminar con este loop.

El mecanismo que una aplicación sigue cuando el usuario quiere terminar con ella es:

1. Windows envia un mensaje (WM_CLOSE) a la Ventana Principal de la aplicación.

2. El objeto Ventana Principal responde llamando a la función miembro CloseWindow, la cual cierra la Ventana Principal pero antes ve si esto es posible llamando a la función miembro

CanClose(). Este mecanismo es comparado con la frase "Si hay alguien que sepa que esta aplicación no deba ser cerrada, que hable ahora o calle para siempre".

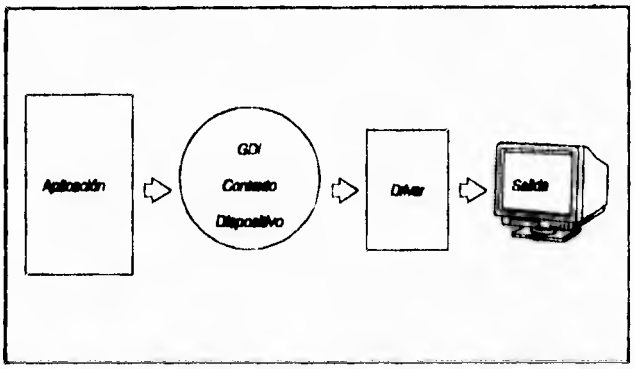
3. La función CanClose() determina si puede o no ser cerrada la ventana.

4. Si todas las ventanas devuelven un valor positivo en la llamada a la función CanClose(), el objeto ventana principal es cerrado y con ello la aplicación termina. En caso de que la función CanClose() devuelva un valor de cero, el objeto principal ventana no es cerrado y la aplicación continuará.

### 4.8.2 Contextos de dispositivo.

Una de las estructuras fundamentales que se utilizan en las funciones API Windows es el contexto de dispositivo(DC). Un contexto de dispositivo es la conexión entre una Aplicación Windows, un controlador de dispositivo y un dispositivo de salida, por ejemplo una pantalla o impresora. Cada programa en Windows puede emplear las funciones GDI para hacer una salida en un dispositivo de salida. La parte GDI de Windows pasa las llamadas que son independientes del dispositivo al controlador del dispositivo. Este controlador tiene la tarea de convertir estas llamadas GDI en operaciones dependientes del dispositivo.

Gráficamente:




El contexto de dispositivo contiene diversas variables que las funciones de la interfaz de dispositivos gráficos (GDI) emplean para dibujar en una superficie abstracta vinculada a una ventana en particular. Algunos de los atributos que se controlan en un dispositivo de contexto son:

- Color de fondo y modo
- Tipo de pincel y color
- Paleta de colores
- Tipo de pluma, color y posición
- Colores fuente y para un texto
- Coordenadas de una ventana

Cada ventana que se crea cuenta con su propio contexto de dispositivo. Para emplear alguna función GDI, se tiene que pasar a dicha función un manejador de contexto de dispositivo. Para obtener este manejador se utiliza esta funcion:

```
HDC DC = GetDC(HWindow)
```

Donde HWindow es la ventana de la cual se obtendrá su contexto de dispositivo. Al terminar de usar las funciones GDI se deberá liberar este contexto con la función

```
ReleaseDC(DC, HWindow)
```

### 4.8.3 Procesamiento de Mensajes.

Los programas de ObjectWindows tienen dos canales de comunicación con Windows. Por un lado, la aplicación controla la interfase con el usuario al llamadar funciones de Windows. Asimismo, Windows envía mensaje a la aplicación en respuesta a algún evento. Como cuando el usario selecciona el item de un menú, o da un doble click con el mouse.

La combinación de los mensajes y funciones constituyen el esqueleto de una aplicación ObjectWindows. El trabajo primordial de la aplicación consiste en responder a los eventos de Windows. Para cada mensaje que se quiera procesar se deberá escribir una función. A estas funciones se les llama funciones de respuesta a mensajes.

**Tipos de mensajes Windows:**

- Manejo de ventanas. Indican que el estado de una ventana ha cambiado. Por ejemplo: WM_CLOSE es enviado cuando una ventana requiere ser cerrada.

- De Inicialización. Enviados cuando es creada una ventana o una caja de diálogo. Como: WM_CREATE.

• Entrada. Es recibido como resultado de una entrada a través del ratón, teclado, barra de scroll.

• Sistema. Enviados en respuesta a la manipulación de los accesorios estándar de la aplicación por parte del usuario. Ejemplo: Presionar el boton de maximizar de una ventana.

• Clipboard. Se generan cuando una aplicación trata de tomar datos provenientes del Clipboard.

• Información del sistema. Es enviado cuando cambia algún atributo del sistema, como el color de fondo.

• Manejo de controles. Son enviados de la aplicación a sus controles, como en el caso de las listbox.

• Notificación de control. Informan a la ventana padre que un evento ha ocurrido, tal como la selección de un item de una caja.

• Scroll-bar. Son generados al modificar la posición del scroll.

• Area no cliente. Mensajes provenientes fuera del área cliente de una ventana.

• Interfase múltiple de documentos. Estos mensajes son enviados por la aplicación que sigue el estándar MDI para controlar las ventanas hijos.

Ejemplo donde se manejan algunos mensajes:

```
#include <owl.h>

class TInicio : public TApplication
{
 public:
  TInicio(LPSTR AName, HANDLE hInstance, HANDLE hPrevInstance,
        LPSTR lpCmdLine, int nCmdShow) : TApplication(AName,
           hInstance, hPrevInstance, lpCmdLine, nCmdShow) { };
   virtual void InitMainWindow();
}

class TVentana : public TWindow
{
 public:
   TVentana(PTWindowObject AParent,LPSTR Title):
          TWindow(AParent, Title);
   virtual void WLButtonDown(RTMessage Msg)
           = [WM_FIRST + WM_LBUTTONDOWN];
   virtual void WLButtonDblClk(RTMessage Msg)
           = [WM_FIRST + WM_LBUTTONDBLCLK];
}

TVentana::TVentana(PTWindowsObject AParent, LPSTR ATitle) :
      TWindow(Aparent, ATitle)
{
  Attr.X = 100;
  Attr.Y = 200;
  Attr.W = 400;
  Attr.H = 180;
}
```

```
void TVentana::WLButtonDown(RTMessage Msg)
{
     HDC Contexto = GetDC(HWindow);
     TextOut(100,100,"presionó el botón izquierdo del mouse",38);
     ReleaseDC(Contexto,HWindow);
}

void TVentana::WLButtonDblClk(RTMessage Msg)
{
     HDC Contexto = GetDC(HWindow);
     char coordenadas[40];
     sprintf(coordenadas,"Coordenadas:(%d,%d)",
             Msg.LP.Lo,Msg.Lp.Hi);
     TextOut(100,100,coordenadas,strlen(coordenadas);
     ReleaseDC(Contexto,HWindow);
}

void TInicio::InitMainWindow()
{
     MainWindow = new TVentana(NULL,"Aplicacion Uno");
}


int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR
                   lpmCmdLine, int nCmdShow)
{
     TInicio Aplica("Aplicacion1", hInstance, hPrevInstance,
                    lpCmdLine, nCmdShow);
      Aplica.Run():
      return Aplica.Status;
}
```

Analizando cada parte:

```
#include <owl.h>
```

▸ Se define una clase derivada de TApplication

```
class TInicio : public TApplication
{
 public:
  TInicio(LPSTR AName, HANDLE hInstance, HANDLE hPrevInstance,
         LPSTR lpCmdLine, int nCmdShow) : TApplication(AName,
            hInstance, hPrevInstance, lpCmdLine, nCmdShow) { };
 virtual void InitMainWindow();
}
```

▸ Se define una clase para la Ventana Principal.

```
class TVentana : public TWindow
{
 public:
  TVentana(PTWindowObject AParent, LPSTR Title) :
              TWindow(AParent, Title):
  virtual void WLButtonDown(RTMessage Msg)
    = [WM_FIRST + WM_LBUTTONDOWN];
  virtual void WLButtonDblClk(RTMessage Msg)
    = [WM_FIRST + WM_LBUTTONDBLCLK];
}
```

▸ Constructor de la Ventana Principal

```
TVentana::TVentana(PTWindowsObject AParent, LPSTR ATitle) :
      TWindow(Aparent, ATitle)
{
  Attr.X = 100;
  Attr.Y = 200;
  Attr.W = 400; // ancho
  Attr.H = 180; // largo
}
```

▸ Se define la funcion WLButtonDown() la cual pertenece a la Ventana Principal, se llamará a esta funcion en forma automática cuando el usuario presione el botón izquierdo del mouse.

```
void TVentana::WLButtonDown(RTMessage Msg)
{
    HDC Contexto = GetDC(HWindow);
  TextOut(100,100,"presionó el botón izquierdo del mouse",38);
    ReleaseDC(Contexto,HWindow);
}
```

▸ Se define la funcion WLButtonDblClk() la cual pertenece a la Ventana Principal, se llamará a esta funcion en forma automática cuando el usuario oprima dos veces el botón izquierdo del mouse.

```
void TVentana::WLButtonDblClk(RTMessage Msg)
{
  HDC Contexto = GetDC(HWindow);
  char coordenadas[40];
  sprintf(coordenadas,"Coordenas : (%d,%d)",Msg.LP.Lo,Msg.Lp.Hi);
 TextOut(100,100,coordenadas,strlen(coordenadas);
  ReleaseDC(Contexto,HWindow);
}
```

▸ Se construye el objeto Ventana Principal, el cual es almacenado en el dato miembro MainWindow del objeto aplicación.

```
void TInicio::InitMainWindow()
{
  MainWindow = new TVentana(NULL,"Aplicacion Uno");
}
```

▸ Se llama a la función miembro Run() de la aplicación, la cual llamará a las funciones InitApplication() e InitInstance() para ejecutar las inicializacion de la primera instancia y de cada instancia. Después es llamada la función InitMainWindow() para crear la ventana principal. Se coloca la aplicación en movimiento por medio de la llamada a MessageLoop() para comenzar a procesar los mensajes de Windows. Además se regresa el status final de la aplicación.

```
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR
                   lpmCmdLine, int nCmdShow)
{
     TInicio Aplica("Aplicacion1", hInstance, hPrevInstance,
                    lpCmdLine, nCmdShow);
     Aplica.Run():
     return Aplica.Status;
}
```

# CAPITULO 5. ANALISIS Y DISEÑO DEL SISTEMA.

El usuario tiene la necesidad de contar con una herramienta mediante la cual pueda crear una presentación gráfica, es decir mostrar mediante una secuencia de imágenes gráficas (ya creadas) el desarrollo de un proyecto, actividad, historia, etc.

Se requiere que el presentador gráfico sea capaz de conducir y modificar la secuencia de imágenes por medio de pulsaciones de teclas o con base en el tiempo transcurrido. Al momento de mostrar la imagen se desea que aparezca con cierto efecto.

Para poder llevar el control del despliegue de las imágenes gráficas de un tema en particular se requiere que las imágenes estén asociadas a ese tema. Por tanto, al seleccionar algún tema del presentador gráfico se pretende que:

- Se muestren una secuencia de imágenes.
- O bien, aparezcan los subtemas relacionados con ese punto en particular.

Asimismo se necesita que los temas e imágenes definidos por el usuario sean agrupados bajo un mismo nombre con el propósito de que al efectuar alguna operación sobre la presentación, (como Salvar o Abrir) ésta se lleve a cabo sobre todo el conjunto.

## 5.1 Objetivos del Sistema:

▸ Proporcionar al usuario un medio mediante el cual pueda definir los temas(opciones) que podrán ser seleccionadas durante la presentación gráfica.

▸ Poder asociar cada tema, contenido dentro de la presentación, con:
- Una Imagen.
- Un conjunto de Imágenes.
- Otros subtemas.

▸ Desplegar las imágenes por un determinado período de tiempo o al momento de ocurrir un determinado evento y bajo cierto efecto.

▸ Agrupar los temas e imágenes definidos bajo un mismo nombre.

## 5.2 Análisis de la Estructura del Objeto.

Se ocupa de los tipos de objetos y sus asociaciones. Estos tipos de objetos guiarán al diseñador en la definición de las clases y sus estructuras de datos.

Dentro de esta etapa se tendrá que identificar lo siguiente:

- Tipos de objetos y las asociaciones entre ellos.
- Supertipos y subtipos de objetos.
- Composición de objetos.

El Sistema requerido consta básicamente de dos partes:

**1. Captura.** En esta parte se definirán los temas de la presentación gráfica, imagenes a cargar, efectos, tiempo que estarán la imagenes en la pantalla.

**2. Presentación.** Corresponde a mostrar los temas de la presentación y al despliegue de la secuencia de imágenes asociadas. Ambos se encuentran definidos en la parte de Captura.

Se pensó que la mejor forma para que el usuario pudiera definir y correr la presentación gráfica era con el uso de ventanas por las siguientes razones:

- Se pueden tener varias ventanas abiertas al mismo tiempo.
- Es posible saltar de una a otra para activar aquella con la que se desea trabajar.
- Pueden ser agrupadas en conjunto con el fin de integrar la presentación.

**5.2.1 Se reconocieron las siguientes Categorías de Objetos:**

**1. Ventana Control.** Encargada de administrar las ventanas que se generen y de servir de liga entre la parte de Captura y Presentación.

**2. Ventana Menu.** En ella se definirán los temas de la presentación. Estos temas conformarán las opciones del Menú. Cada opción será asociada a una Imagen, un conjunto de Imágenes, o bien a otro Menú.

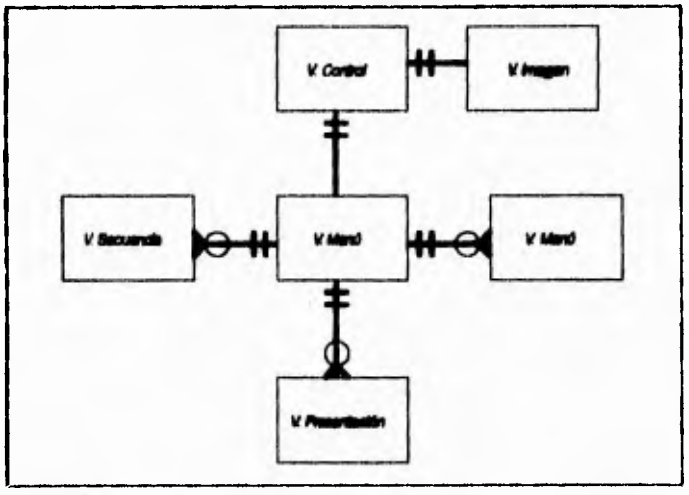
**3. Ventana Secuencia.** Aqui se colocará la ubicación física de las imágenes a cargar asociadas a algún tema.

**4. Ventana Presentación.** Contendrá la ubicación física de la Imagen a cargar asociada a algún tema.

**5. Ventana Imagen.** Sobre ella serán desplegadas una por una la secuencia de imágenes definida.

### 5.2.2 Asociaciones entre tipos de Objetos.

Una vez reconocidos los tipos de objetos que existen, se identificó que están asociados de la siguiente forma:



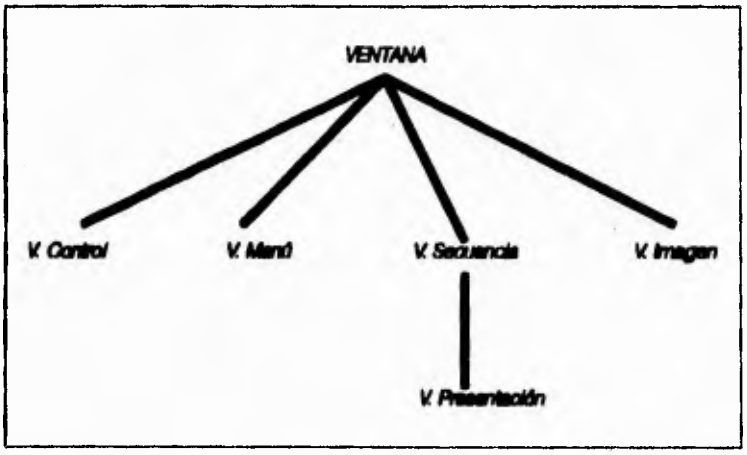
La Ventana Control será la encargada de administrar todas las Ventanas y de servir de liga entre la parte de Captura y Presentación.

Las Ventanas de Menú, Secuencia y Presentación serán los tipos de objetos utilizados para la parte de Captura. Cada Ventana Menú podrá ser asociada desde cero a muchas Ventanas de Secuencia, Presentación o Menú.

La Ventana Imagen será la encargada de desplegar una a una las imágenes capturadas.

### 5.2.3 Supertipos y subtipos de objetos.

Indicarán las relaciones de herencia entre las clases. Las categorías de objetos identificados anteriormente, se agruparon en un árbol de herencia que se presenta a continuación:



## 5.3 Análisis del Comportamiento del Objeto.

Se ocupa de modelar lo que ocurre a los objetos al paso del tiempo.

Se deberá identificar lo siguiente:

1. Estados que puede tener un objeto.
2. Eventos que ocurren.
3. Operaciones que se llevan a cabo.

Mediante esquemas de eventos podemos describir los procesos en términos de eventos, de reglas de activación y operaciones mostrando los distintos estados por los que pasa la aplicación.

Diagrama de eventos para la parte de Captura.

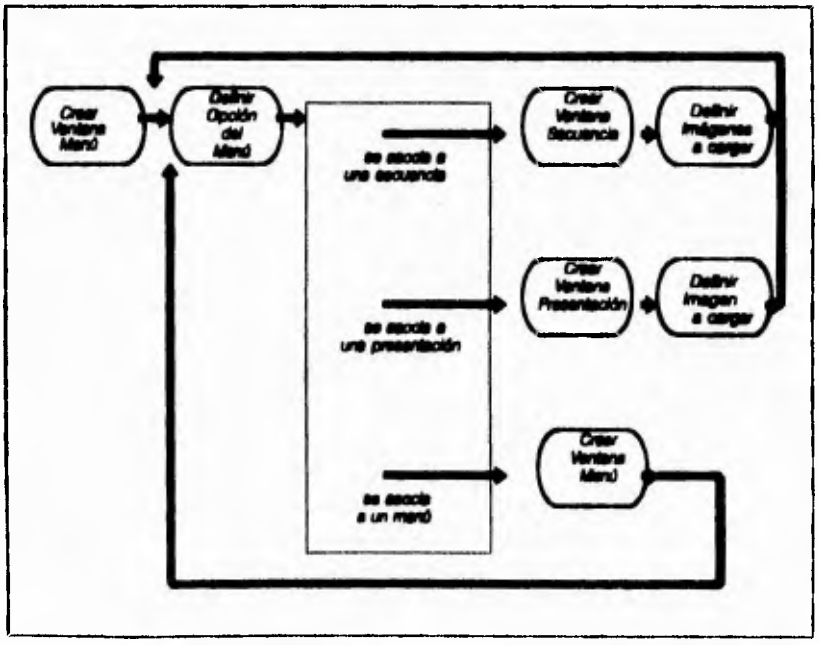
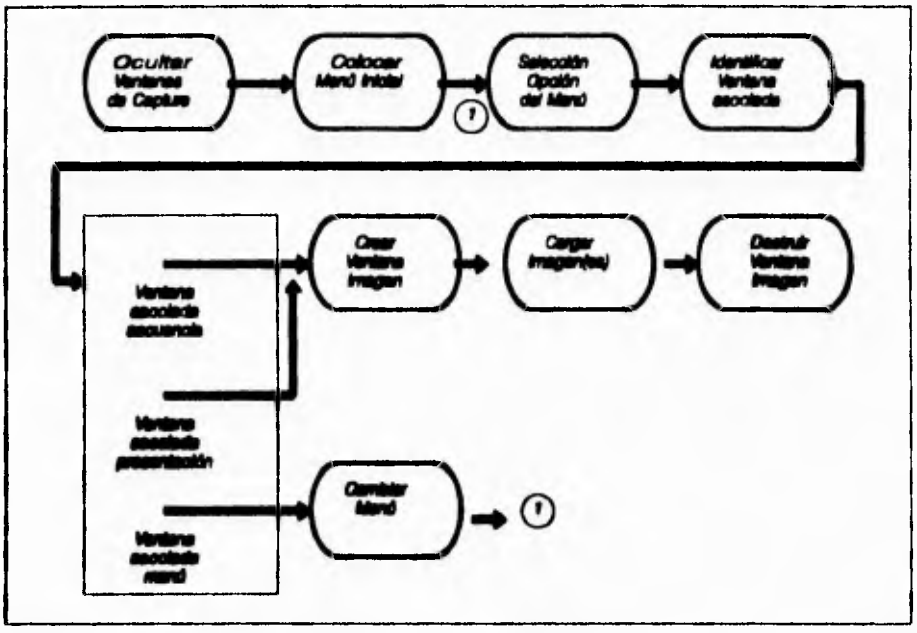


Diagrama de eventos para la parte de Presentación.




## 5.4 Diseño de la Estructura y Comportamiento del Objeto.

En esta etapa se deberá identificar:

- Clases a implementar. Los tipos de objetos identificados en el **Análisis** de la Estructura del Objeto serán la guía para esta decisión.

- Estructura de datos que utlizará cada clase.

- Operaciones que ofrecerá cada clase. Se deberán definir tomando como base los procesos obtenidos del Análisis del Comportamiento del Objeto.

- Herencia, Composición.

Se pensó que la mejor opción para poder cubrir los requerimientos del sistema era mediante el uso de la Interfaz de Documentos Múltiple (MDI).

Ya que MDI permite al usuario trabajar simultáneamente con diferentes documentos. Cada documento aparece en su ventana propia, se pueden tener abiertas varias ventanas al mismo tiempo, y se puede saltar de una a otra para poder activar el documento con el cual se desea trabajar.

Las ventanas Marco y Cliente de la Interfaz Múltiple de Documentos cubren los requerimientos de la ventana Control. Esto es, mediante estas dos ventanas podemos administrar y controlar cualquier tipo de ventana asociadas a ella tal como: Menú, Secuencia, Presentación e Imagen.

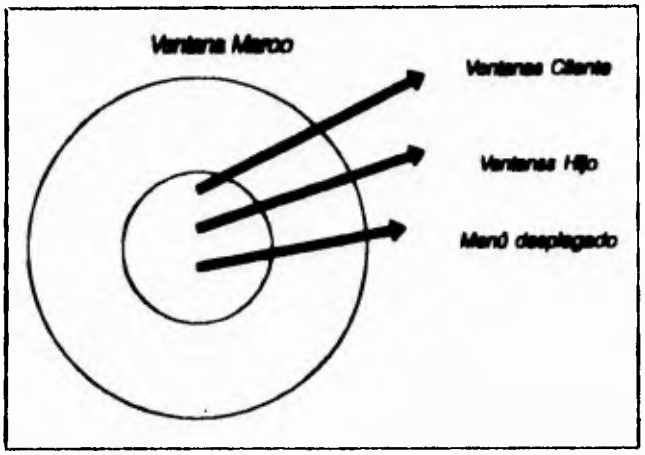
Las ventanas asociadas no pueden poseer un menú propio, así que sus funciones son controladas por el menú de la ventana Marco. Con esto es posible realizar operaciones sobre todas las ventanas generadas en la aplicación.

Todos los documentos (ventanas) generados por el usario serán agrupados bajo un mismo nombre al que se le llamará PROYECTO.
Las operaciones de: Salvar, Salvar Como, Abrir, Cerrar, Nuevo, etc tendrán efecto sobre todo el PROYECTO.

## 5.5 CLASES A IMPLEMENTAR:

### 1. Ventana Marco.

Estructura:




Datos:

- **Ventanas Creadas**. Manejadores de las ventanas asociadas.
- **Ventana cliente**. Manejador de la ventana cliente.
- **Menú desplegado**. Identificador del menú desplegado.

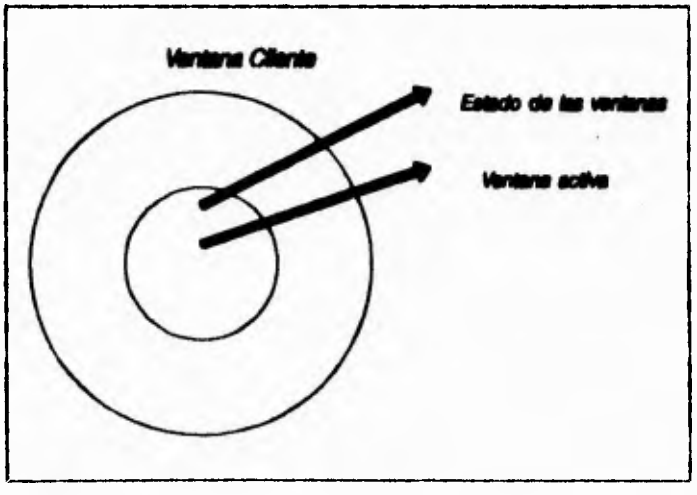
Operaciones:

- **Constructor**. Crea la ventana Marco.
- **Crea ventana cliente**. Crea ventana sobre la cual aparecerán las ventanas asociadas.
- Crea ventana inicial. Crea la primera ventana de captura.

- Crea ventana Menú. Crea ventana para definir temas de la presentación.
- Crea ventana Secuencia. Crea ventana para definir imágenes.
- Crea ventana Presentación. Crea ventana para colocar imagen.
- Cambia menú. Modifica el menú de la ventana Marco.
- Abrir. Carga una nueva aplicación. (Termina la anterior si es que la hubiera).
- Salvar. Graba aplicación.
- Ejecutar. Inicia la presentación gráfica con base en la captura realizada.
- Prueba. Muestra alguna imagen o menú en particular.
- Nueva. Cerrar ventanas definidas por el usuario y limpiar ventana inicial de captura.
- Salir. Término de la aplicación.

**2. Ventana Cliente.**

Estructura de la ventana:



Datos:

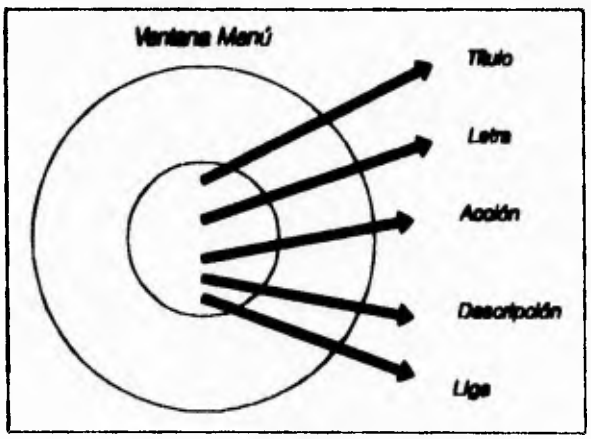
- Estado de las ventanas. Identifica la situación en la cual están las ventanas. (Oculta, maximizada, iconizada, etc).
- Ventana Activa. Manejador de la ventana activa.

Operaciones:

- Constructor. Crea la ventana cliente.
- Pinta. Coloca las marcas de captura.
- Oculta ventanas. Desaparece las ventanas de captura para iniciar presentación.
- Muestra ventanas. Aparece las ventanas de captura.
- Activa ventana. Desactiva la ventana que tiene el foco y manda activar la ventana deseada.

**3. Ventana Menu.**

Estructura:



Datos:

- Título. Es el tema de la presentación, el cual será representado mediante una opción del Menú.
- Letra. Acelerador del Menú. Es una letra que forma parte del Título. Mediante ésta se podrá seleccionar rápidamente la opción.
- Acción. Indica a que tipo de ventana será asociada la opción del Menú. Puede ser:

Secuencia. Se asocia con un conjunto de imágenes.
Presentación. Se liga con una imágen.
Menú. Se agrupa con un conjunto de opciones (otro Menú).

- Descripción. Referencia que hace el usuario.
- Liga. Muestra la ubicación física en la cual se encuentra la ventana asociada.

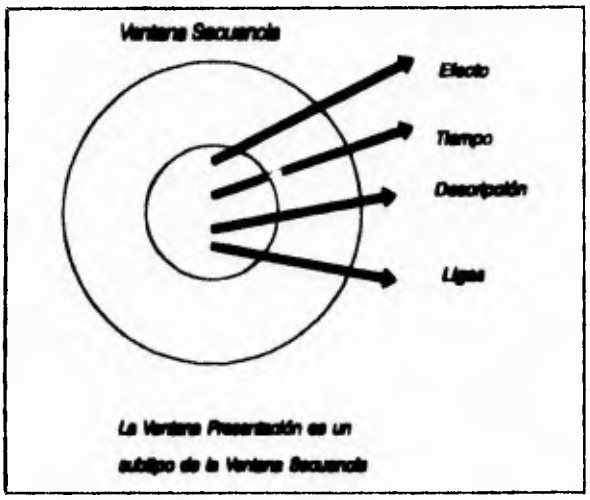
Operaciones:

- Constructor. Crea la ventana menú.
- Abrir. Recupera información contendida en la ventana.
- Cerrar. Destruye la ventana.
- Capturar dato. Ingresa algún dato a la ventana.
- Scroll. Desplaza el contenido de la ventana.
- Paint. Redibuja el contenido de la ventana cuando ésta ha sido modificada.
- Doble click del mouse. Coloca la caja de diálogo para iniciar captura.
- Proporciona datos. Accesa y devuelve datos del objeto.
- Activa ventana. Coloca el foco sobre esta ventana.
- Colocar celdas. Dibuja en la ventana celdas para facilitar la captura de datos.

- Botón izquierdo del mouse presionado. Reconoce que la celda de la ventana se ha seleccionado.

**4. Ventana Secuencia.**

Descripción de campos:




**Datos:**

- Efecto. Indicará la forma en la cual será desplegada la imagen.
- Tiempo. Indicará los segundos que la Imagen permanecerá desplegada, la ausencia de tiempo implicará el uso de alguna tecla.
- Descipción. Título de la Imagen.
- Ligas. Ubicación física de la Imágenes a cargar.

Operaciones:

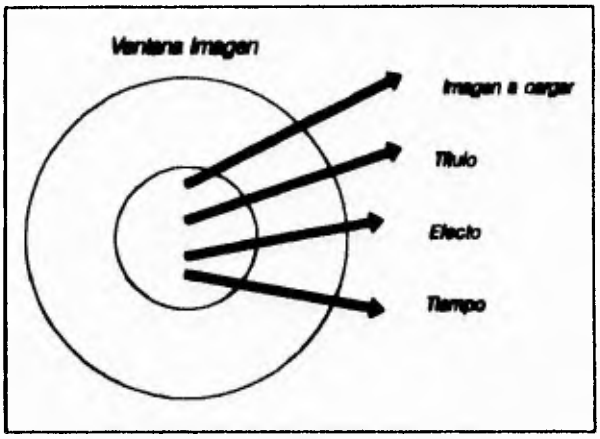
- Constructor. Crea la ventana secuencia.
- Abrir. Recupera información contendida en la ventana.
- Cerrar. Destruye la ventana.
- Capturar dato. Ingresa algún dato a la ventana.
- Scroll. Desplaza el contenido de la ventana.
- Paint. Redibuja el contenido de la ventana cuando ésta ha sido modificada.
- Doble click del mouse. Coloca la caja de diálogo para iniciar captura.
- Proporciona datos. Accesa y devuelve datos del objeto.
- Activa ventana. Coloca el foco sobre esta ventana.
- Colocar celdas. Dibuja en la ventana celdas para facilitar la captura de datos.
- Botón izquierdo del mouse presionado. Reconoce que la celda de la ventana se ha seleccionado.

**5. Ventana Presentación.**

Se identificó que es un subtipo de la Ventana Secuencia, con ello todo lo que se aplicó a la Ventana Secuencia tendrá efecto sobre la Ventana Presentación. La diferencia radica en que solo se definirá una imagen en lugar de un conjunto de ellas.

**6. Ventana Imagen.**

Descripción de campos:




Datos:

- Imagen a cargar. Ubicación física de la imagen a desplegar.
- Título. Nombre de la imagen.
- Efecto. Forma mediante la cual será desplegada la imagen.
- Tiempo. Período en el cual será desplegada la imagen.

Operaciones:

- Constructor. Crea la ventana imagen.
- Obten datos. Busca la imagen a cargar, tiempo, efecto, título con base en el contenido de las ventanas de captura.
- Carga imagen. Despliega una imagen bitmap.
- Coloca efecto. Al momento de desplegar la imagen muestra un determinado efecto.
- Paint. Redibuja el contenido de la ventana cuando ésta ha sido modificada.

- Calcula tiempo. Tiempo que permanecerá la imagen desplegada.
- Borra imagen. Destruye la imagen de la ventana.
- Cerrar. Destruye la ventana.
- Maneja pulsaciones del teclado. Modificarán la secuencia de la imagen bitmap desplegada con base en lo siguiente:

Inicio. Muestra la primera imagen.
Página arriba. Carga la imagen anterior.
Página abajo. Despliega la imagen siguiente.
Enter. Carga la imagen siguiente.
Fin. Muestra la última imagen.
Escape. Termina la secuencia de imagenes, destruye ventana de presentación y muestra el menú previo.

Para poder guardar la presentación gráfica se asoció cada ventana con un archivo.

Se manejan 4 tipos diferentes de extensiones para los archivos:

*.pry Identifica a la primera ventana de captura, es de tipo Menú, y será la que guie el desarrollo del proyecto.

*.men Para definir temas de la presentación. Es del tipo Menú.

*.sec En ella se colocan las imágenes asociadas a algún tema.

*.pre Aqui se pone la imágen asociada a algún tema.

El tipo de imágenes requeridas por el usuario para ser manejadas por la aplicación son del tipo Bitmap. En el **apéndice A** se presentan las características principales de este tipo de imagen.

El lenguaje que se escogió para desarollar el Sistema fue Borland C++ por las siguientes razones:

- Ser un lenguaje portable, flexible y eficiente.

- No imponer una sobrecarga de ejecución demasiada alta.

- Conocimiento previo del lenguaje C. C++ es una extensión orientada a objetos del lenguaje C.

- Contar con el compilador de Borland C++ 3.1 al momento de iniciar la tesis.

# CONCLUSIONES.

El estado actual de la ingeniería del software está atrasado con respecto a las demas áreas de la ingeniería. Cuando la mayoría de los ingenieros completan su trabajo y éste se vende, se espera que funcione de manera correcta. No se espera que los motores de un avión exploten o que los edificios se derrumben, pero no nos sorprende que cierto software funcione de manera inadecuada. La mayoría de los productos de hardware tienen garantía, pero la mayoría de los productos de software llevan una renuncía a ella. El software se vende no cuando está libre de errores, sino cuando éstos aparecen con frecuencias bajas.

Desde la revolución industrial, la fabricación de técnicas ha evolucionado en forma extraordinaria, de herramientas artesanales a herramientas poderosas; de la producción en serie a las fábricas robóticas flexibles. La construcción del software también debe rebasar la etapa de los métodos artesanales.

La mayoría del software de la actualidad está diseñado y codificado con técnicas manuales; Cada programa es una pieza artesanal única, un poco como la hechura de ropa en los telares antes de la revolución industrial o la fabricación de armas por armeros individuales en el siglo XVIII.

Una de las preocupaciones más importantes en la industria de la computación es la de crear un software complejo, flexible, rápido de desarrollar, confiable, barato, fácil de mantener.
Se necesita construir un software tan complejo y confiable como los jumbo jets o la red telefónica mundial y tan rico como una biblioteca judicial o los archivos de un historiador de arte.

Con el uso de las técnicas orientadas a objetos se logrará una revolución industrial en el software ya que éste será logrado a partir de componentes de objetos reutilizables. Se pasará de una era de paquetes monolíticos de software donde un vendedor "construye" todo un paquete, hasta una era en la que el software será ensamblado a partir de componentes y paquetes de muchos proveedores (de la misma forma en que las computadoras y los automóviles son ensamblados a partir de muchos proveedores). Los componentes serán cada vez más complejos desde el punto de vista interno, pero será más sencillo interactuar con ellos. Se construirán tipos de objetos a partir de objetos más sencillos. Una vez que los tipos de objetos funcionen bien, el diseñador los considerará como cajas negras, de modo que nadie pueda ver en su interior. La ingeniería del software adquirirá de esta forma más características de la ingeniería del hardware.

Con la orientación a objetos cambiará nuestra forma de pensar sobre los sistemas. Para la mayoría de las personas la forma de pensar en objetos es más natural que las técnicas del análisis y diseño estructurado. Después de todo el mundo está formado por objetos. Se comienza a aprender sobre ellos en la infancia y se descubre que tienen determinado tipo de comportamiento. Los usuarios finales piensan de manera natural en términos de objetos, eventos y mecanismos de activación. Se pueden crear diagramas orientados a objetos que les parezcan familiares, mientras tengan quizá dificultades con los digramas entidad-relación, de flujo de datos, etc.

A las personas que se encargan del desarrollo de un sistema se les ha enseñado a pensar como computadoras. El análisis orientado a objetos nos recuerda, de manera natural, como categorizamos y comprendemos el mundo. Cuanto más complejas sean las computadoras, menos deben pensar los humanos como ellas. En vez de ello, hay que

hacer que las computadoras piensen como humanos.

De esta forma se podrá construir mejores sistemas y se podrá mejorar la comunicación con los usuarios.

# APENDICE A. BITMAPS

A partir de la versión 3.0 de Windows existen dos tipos de Bitmaps:

- Bitmaps dependientes del dispositivo.
- Bitmaps independientes del dispositivo.

Un Bitmap dependiente del dispositivo se compone de un patrón de bits que está depositado en la memoria, y que se puede visualizar con determinadas funciones en un dispositivo de salida. Existe una estrecha relación entre los bits en la memoria y los pixeles del dispositivo. Su estructura se diferencia por ello según el adaptador gráfico empleado.

La dependencia de dispositivo Bitmap lleva a problemas cuando los gráficos se han de pasar de una computadora a otra.

Un Bitmap independiente de dispositivo decsribe el aspecto de la imagen y no un formato específico de dispositivo. Con ayuda de determinadas funciones se convierte la descripción de la imagen independiente del dispositivo al formato de salida actual del dispositivo.

Los Bitmaps en color pueden estar organizados en dos formas:

En la primera posibilidad existen varios niveles (planos) que representan cada uno un color determinado. El color del pixel a visualizar se define en cada nivel mediante un bit. Los niveles están depositados secuencialmente en la memoria. Si un dispositivo de salida puede representar por ejemplo ocho colores, pueden existir tres niveles para ello. Uno para el rojo, otro para el

verde y uno para el azul. Un pixel amarillo se genera activando el bit correspondiente en el nivel azul y en el nivel verde. En este método los parámetros para el número de niveles de color han de recibir el valor de tres, y los del número de bits por pixel el valor de uno.

En la otra posibilidad sólo existe un nivel de color. Los colores se representan mediante varios bits en este nivel. Para ocho colores se necesitan tres bits, es decir, cada pixel es representado por tres bits. En este caso el número de niveles de color es uno y el número de bits por pixel es tres.

Estructura de un archivo Bitmap dependiente del dispositivo:

| | | |
|---|---|---|
| bmType | WORD | 0 |
| bmWidth | WORD | Ancho en pixeles |
| bmHight | WORD | Alto en pixeles |
| bmWidthBytes | WORD | Número de Bytes en una línea |
| bmPlanes | BYTE | Número de niveles de color |
| bmBitsPixel | BYTE | Número de bits por pixel |
| bmBits | | Arreglo para el patrón de bits |

Estructura de un archivo Bitmap independiente del dispositivo:

- Encabezado de archivo.
- Encabezado del Bitmap.
- Tabla de colores.
- Arreglo para el patrón de bits.

1. Encabezado de archivo.

Contiene información acerca del tipo, tamaño y distribución del archivo Bitmap.

```
bfType        WORD          BM
bfSize        DWORD         Tamaño del archivo
bfReserved    WORD          0
bfReserved    WORD          0
bfOffbits     DWORD         Offset al patrón de bits
biSize        DWORD         Tamaño del encabezado del Bitmap
biWidth       DWORD         Ancho en pixeles
biHeight      DWORD         Alto en pixeles
biPlanes      DWORD         Número de niveles de color: 1
bibitCount    WORD        Número de bits por pixel:
                            1,4,8,24.
```

La variable bibitCount indica cuantos bits se necesitan para representar un pixel y está en estrecha relación con el número de entradas en la tabla de colores (la cual es un arreglo de 2 a 256 entradas).

| Bits/pixel | Entradas en la tabla | Descripción |
|---|---|---|
| 1 | 2 | Bitmap monocromático |
| 4 | 16 | Bitmap con 16 colores como máximo. Cada pixel se representa mediante un índice de 4 bits en la tabla. |
| 8 | 256 | Bitmap con 256 colores como máximo. Cada pixel se represnta medianre un índice de 8 bits en la tabla. |
| 24 | NULL | Bitmap con 16M de colores como máximo. Tres bytes en el arreglo del Bitmap representan la intensidad de rojo, verde y azul de un pixel. |

**2.** Encabezado del Bitmap.

Especifica las dimensiones, tipo de compresión y formato de los colores del Bitmap.

| | | |
|---|---|---|
| biStyle | DWORD | Tipo de compresión |
| biSizeImage | DWORD | Tamaño del Bitmap comprimido |
| biXPelsPermeter | DWORD | Resolución horizontal en pixeles del dispositivo por metro |
| biYPelsPermeter | DWORD | Resolución vertical en pixeles del dispositivo por metro |
| biClrUsed | DWORD | Número de índices de color utilizados por el Bitmap |
| biClrImportant | DWORD | Número de índices de color importantes |

**3.** Tabla de colores.

Contiene tantos elementos como colores haya en el Bitmap. Los colores deben aparecer en orden de importancia. Esto ayudará a desplegar el Bitmap cuando el dispositivo no soporte todos los colores requeridos.

| | | |
|---|---|---|
| rgbBlue | BYTE | Componente azul del color |
| rgbGreen | BYTE | Componente verde del color |
| rgbRed | BYTE | Componente rojo del color |
| rgbReserved | BYTE | 0 |

**4.** Arreglo de bytes que representan el Bitmap.

El primer byte en el arreglo representa los pixeles en la esquina inferior izquierda del Bitmap y el último byte representa los pixeles en la esquina superior derecha del Bitmap.

**Compresión de Bitmaps.**

Desde la versióm Windows 3.0 se utiliza la compresión con el fin de reducir la cantidad de memoria y disco requerida para el Bitmap.

▸ Compresión de 8 bits por pixel.

Este formato tiene dos modos:

1. Modo Encoded.
2. Modo Absoluto.

Pueden estar presentes ambos modos según convenga a la compresión de la imagen.

**1.** Modo Encoded.

La unidad de información en este modo consiste en dos bytes. El primer byte especifica el número consecutivo de pixeles a ser dibujados utilizando el índice de color contenido en el segundo byte.

| Datos Comprimidos. | Datos Expandidos. |
|---|---|
| 03 04 | 04 04 04 |

El primer byte del par puede ser 0 lo cual indicará que se trata del fin de una línea, bitmap o un desplazamiento.

La interpretación de este byte depende del valor del segundo byte del par, el cual debe estar en un rango de 0x00 a 0x02.

| Segundo byte. | Significado |
|---|---|
| 0 | Fin línea |
| 1 | Fin Bitmap |
| 2 | Los bytes siguientes indicarán un desplazamiento horizontal y vertical para localizar el pixel siguiente. |

| Datos Comprimidos. | Datos Expandidos. |
|---|---|
| 00 02 05 01 | Mueve 5 a la derecha y 1 hacia abajo para localizar siguiente pixel. |

**2.** Modo Absoluto.

El primer byte es puesto en cero, el segundo se coloca con un valor entre 0x03 y 0xFF. El segundo byte representa el # de bytes que siguen, cada uno de los cuales contienen el índice de color de un pixel.

| Datos Comprimidos. | Datos Expandidos. |
|---|---|
| 00 03 45 56 67 | 45 56 67 |

▸ Compresión de 4 bits por pixel.

Este formato tiene dos modos:

1. Modo Encoded.
2. Modo Absoluto.

Pueden estar presentes ambos modos según convenga a la compresión de la imagen.

**1.** Modo Encoded.

La unidad de información en este modo consiste en dos bytes. El primer byte del par contiene el número de pixeles a dibujar utlizando los índices de color del segundo byte.

El segundo byte contiene dos índices de color. Uno en el nibble alto (cuatro primeros bits) y otro en el nibble bajo (cuatro últimos bits). El primer pixel es dibujado usando el color especificado en el nibble alto, el segundo pixel es dibujado usando el color indicado en el nibble bajo, el tercero es desplegado con el color indicado en el nibble alto y asi sucesivamente hasta que todos los pixeles especificados en el primer byte hayan sido dibujados.

| Datos Comprimidos. | Datos Expandidos. |
|---|---|
| 03 04 | 0 4 0 |
| 05 26 | 2 6 2 6 2 |

El primer byte del par puede ser colocado en cero para indicar el fin de línea, bitmap o un desplazamiento. La interpretación de la acción a tomar depende del segundo byte del par. Este segundo byte del par tiene un valor de 0x00 a 0x02. Su significado es el mismo de la compresión de ocho bits por pixel.

**2.** Modo Absoluto.

El primer byte del par se coloca en cero, el segundo contiene el número de índices de colores que siguen y los siguientes bytes contienen índices de color en el nibble alto y bàjo, un índice de color para cada pixel.

| Datos Comprimidos. | Datos Expandidos. |
|---|---|
| 00 06 45 56 67 | 4 5 5 6 6 7 |

# APENDICE B. Manual del Usuario

Al ejectuar el programa Sistema de Captura y Presentación Gráfica, se puede visualizar una ventana divida en celdas, dentro de la cual el usuario puede comenzar a estructurar su propio proyecto.

Cada una de las celdas pueden ser seleccionadas haciendo un click con el mouse en su posición. Al hacer doble-click dentro de cada una de éstas, se activarán, y desplegarán diálogos, en los cuales podrá ser introducida la información.

Como la primera ventana es un menú, primero se explicarán cada una de las celdas en que se encuentra dividida: (ver figura B.1)

La primera celda representa el título que llevará cada una de los items del menú creado. El máximo tamaño es de 15 caracteres. Si se rebasa esta cantidad, los caracteres restantes serán ignorados. Esta opción no puede ser omitida.

La segunda celda indica que el item elegido anteriormente se activará a través del mouse, o por medio de la secuencia de teclas ALT + <letra definida en esta celda>, esto es, la celda representa el acelerador. Si la celda se omite, el item solo podrá ser elegido a través del mouse, y si se define una letra que no esté contenida en el nombre del título, este item tampoco contará con un acelerador. Como se dijo anteriormente, si se rebasa el límite de 15 caracteres para el título y se define una letra que cae dentro de los restantes caracteres, entonces el título será desplegado sin acelerador. La tercera celda indica la acción que realizará este item del menú. Entre las acciones tenemos:

Menú, Secuencia, Presentación y Borrar.

Si se elige la opción Borrar, toda la información entrada para este item, se borrará y podrá ser reutilizada nuevamente la fila.

La cuarta celda corresponde a la descripción para el item del menú. Solo sirve como referencia para saber que hace esa opción. Esta celda puede omitirse.

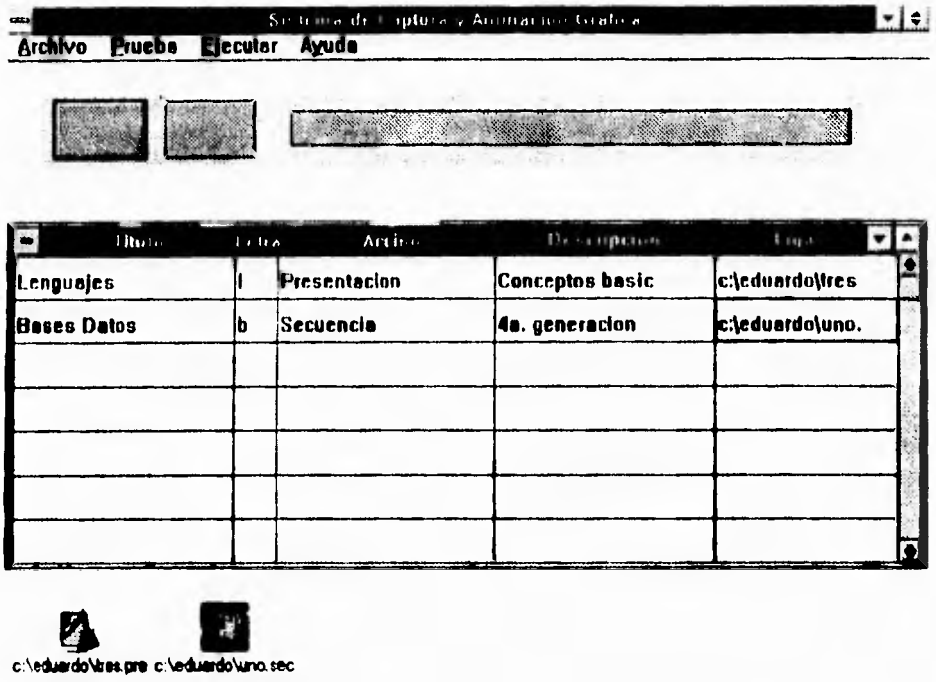
La quinta celda indica la liga, es decir, el tipo de archivo que tendrá asociada la acción elegida. Aquí, el tipo de archivo varía de acuerdo a que acción se haya seleccionado, y son los siguientes:

- Para menú: *.men
- Para secuencia: *.sec
- Para presentación: *.pre

Después de haber indicado el archivo, se crea un icono en la parte inferior de la pantalla que representa a la nueva ventana para la acción indicada. (ver figura B.1)

Para las celdas 4, 5 y 6, la activación de las celdas se hará automáticamente, esto es, después de indicar la acción, de inmediato se desplegará un diálogo preguntando por la descripción y enseguida se activará el de la liga.

**Figura B.1**




Cuando en la acción se eligió:

1.- Menú. significa que se quiere tener un submenú dentro de algún item. En la liga puede indicarse el nombre de un archivo que ya existe o crear uno nuevo. La ventana que se crea para esta acción, tiene las mismas celdas que se describieron con anterioridad.

2.- Secuencia. indica que al elegir este item se desplegará una secuencia de 2 o más imágenes. La ventana que se crea para esta opción contiene las siguientes celdas: (ver imagen B.2) La primera celda indica el efecto con que se despliega una imagen, y puede ser en: cuadros, scroll, pedazos, tamaño.

En la segunda celda se le indica el tiempo en segundos que durará presentada la imagen.

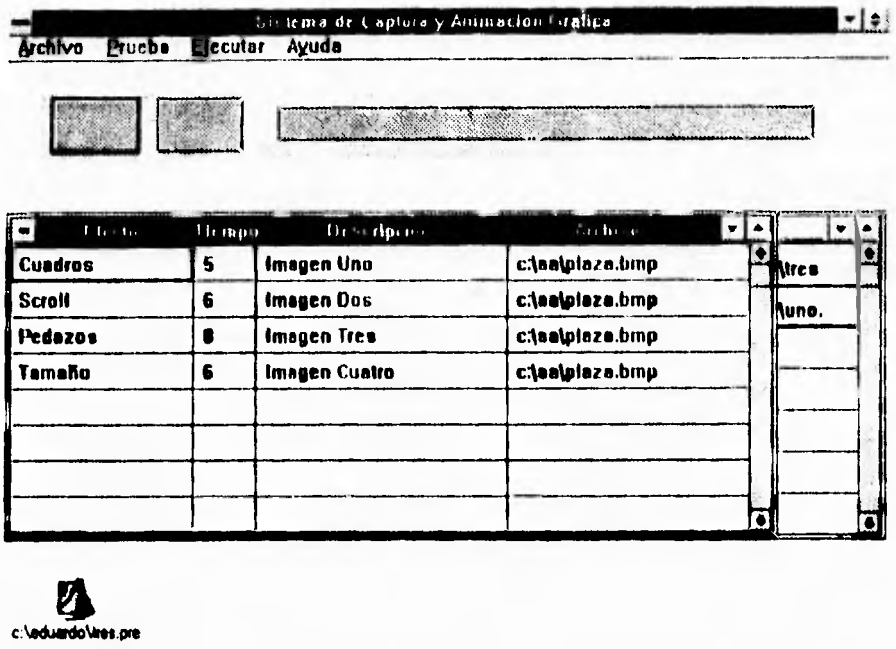
La tercera celda contiene la descripción acerca de la imagen. La descripción dada sirve también para poner ésta como título al momento de desplegar la imagen.

La cuarta celda guarda el nombre y la ubicación del archivo bitmap que será desplegado.

Cuando se va mostrando la secuencia de imágenes, el tiempo que dura cada una depende del tiempo especificado en la celda anteriormente mencionada. Pero también es posible utilizar las siguientes teclas para cambiar el orden de presentación:

- Home. esta tecla nos permite ir a la primera imagen.
- Fin. esta tecla nos permite ir a la última imagen
- PgDn. esta tecla nos permite ver la imagen siguiente
- PgUp. esta tecla nos permite ver la imagen anterior
- Esc. para terminar la secuencia presione esta tecla
- Enter. igual que la tecla PgDn

**Figura B.2**




3.- Presentación. Esta acción es similar a la anterior, pero solo puede contener una sola imagen.

Al momento de que el usuario va creando su proyecto tiene la opción de ver el resultado de lo que está haciendo. Esto puede hacerse con la opción Prueba del menú principal. Esta opción nos muestra solo a partir de la ventana en que nos encontremos.

Una vez que se haya terminado de definir todas las secuencias, presentaciones y menús, basta con elegir la opción Ejecutar del menú principal para ver toda la corrida del proyecto. Para salvar el proyecto, se puede elegir la opción Salvar ó Salvar como y éste se guardará en disco con extensión .pry

Al momento de ejecutar el Sistema de captura, el proyecto guardado con anterioridad podrá ser abierto seleccionando la opción Abrir del menú principal.

El sistema cuenta con una ayuda, que proporciona una descripción general de lo que se puede hacer dentro del sistema. Esta podrá ser accesada a través de la opción Ayuda del menú principal

Para salir del Sistema y regresar a Windows, elija la opción Salir del menú principal, o presione la secuencia de teclas ALT + F4.

# BIBLIOGRAFÍA

Windows 3.1 Intern
Juergen Boer / Irene Bouder
Ed. Abacus

Object Oriented Analysis and Design
James Martin / James J. Odell
Ed. Prentice Hall

Developing Windows Application with Borland C++
James McCord
Ed. SAMS

Mastering Borland C++
Tom Swam
Ed. SAMS

Borland C++ Programación Orientada a Objetos
Ted Faison
Ed. Prentice Hall

ObjectWindows for C++
Users Guide
Borland International

Disertación sobre la POO
J. Carlos Cárdenas
Tesis Facultad de Ingeniería, UNAM

Programación Orientada a Objetos
Norberto Arrieta
Apuntes Facultad de Ingeniería, UNAM