UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

UNIDAD ACADEMICA DE LOS CICLOS PROFESIONAL Y DE POSGRADO DEL COLEGIO DE CIENCIAS Y HUMANIDADES
INSTITUTO DE INVESTIGACIONES EN MATEMATICAS APLICADAS Y SISTEMAS

LABORATORIO GRAFICO BASICO DE PROGRAMACION ORIENTADA A OBJETOS

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A

HECTOR GERARDO PEREZ GONZALEZ

DIRECTOR: DRA. HANNA OKTABA

MEXICO, D. F. MARZO 1996

AGRADECIMIENTO

A la Dra. Hanna Oktaba, Directora de esta tesis, por su gran apoyo y por su dedicación a la enseñanza de las ciencias de la Computación la cual se contagia permitiendo la difusión más amplia del conocimiento.

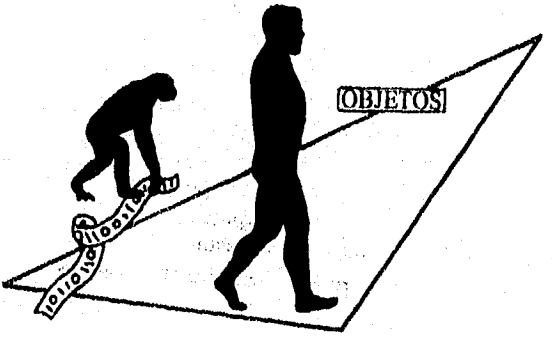
A Rosma, verdadera autora de esta tesis, por su permanente impulso para lograr la exitosa terminación de la misma.

A mis padres y hermanos a quienes agradezco sus palabras de aliento.

A la Universidad Autónoma de San Luis Potosí y mis compañeros de trabajo de la Facultad de Ingeniería.

LABORATORIO GRÁFICO BÁSICO DE PROGRAMACIÓN ORIENTADA OBJETOS

I ANTECEDENTES Y OBJETIVOS



I ANTECEDENTES Y OBJETIVOS

El presente trabajo tiene como objetivo la creación de un sistema automatizado en el que se muestre una representación gráfica de un programa escrito en un lenguaje orientado a objetos. Se propone que éste sistema funcione como un laboratorio que facilite al estudiante de programación, el entendimiento de los conceptos que fundamentan la tecnología orientada a objetos.

Se plantea la realización de un trabajo de investigación que habrá de sustentar al sistema-laboratorio. Los resultados que arroje este trabajo determinarán los alcances y limitaciones de dicho sistema.

INVESTIGACION

La investigación buscará dar respuestas a las siguientes preguntas:

+ Cual lenguaje de programación es el mas adecuado para considerarlo como el objeto de estudio, el cual definirá los conceptos que el sistema ayudará a comprender?

+ Cuales son los alcances y limitaciones que el sistema tendrá?

+ A quién deberá dirigirse el uso del sistema propuesto?

LIMITES TEORICOS

La investigación se limita a analizar la posibilidad de utilizar herramientas para facilitar el proceso de enseñanza-aprendizaje de la programación orientada a objetos. No se dedica demasiada atención a los procesos de análisis y diseño orientado a objetos sino tan sólo a la programación.

LIMITES TEMPORALES

La investigación analiza el periodo de tiempo en el que una filosofía de programación busca madurar y para ello se vale de herramientas de formalización, de especificación y didácticas.

LIMITES ESPACIALES

El trabajo de investigación analiza la situación evolutiva que la enseñanza de la programación orientada a objetos guarda en México.

UNIDADES DE OBSERVACION

Las unidades de observación son principalmente libros y publicaciones acreditadas sobre el tema.

VALOR POTENCIAL DE LA INVESTIGACION

La investigación aportará la aclaración de conceptos relacionados al tema y los diferentes criterios que se toman en cuenta al programar en un lenguaje de programación orientado a objetos en particular.

CONVENIENCIA E IMPLICACIONES PRACTICAS

La investigación servirá como sustento para la creación de un sistema que facilite el aprendizaje de la programación orientada a objetos.

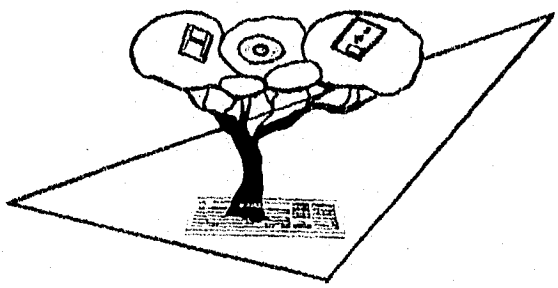
VALOR TEORICO

La investigación ofrece la posibilidad de una exploración fructífera en el campo de los lenguajes de programación y los conceptos en que estos se fundamentan, además presenta información sobre posibles herramientas didácticas que auxilian en la enseñanza de la programación de computadoras.

UTILIDAD METODOLOGICA

Como producto de esta investigación se pueden lograr mejoras en la forma de experimentar con los conceptos que dan fundamento a la tecnología orientada a objetos dentro del ámbito educativo.

II PROGRAMACION ORIENTADA A OBJETOS



II LA PROGRAMACION ORIENTADA A OBJETOS

ANTECEDENTES

El advenimiento de la computadora, como el de todas las creaciones del hombre, se debe principalmente a la inquietud de éste por facilitar, mejorar o acelerar la solución de sus problemas.

Recibir información y procesarla es lo único que básicamente, la máquina es capaz de hacer. Por ello, la solución de cualquier problema debe reducirse a la identificación de los datos en él involucrados y a la de las acciones que con ellos se realizan.

La evolución en la forma de construir programas para que las computadoras efectúen lo que esperamos de ellas, ha conducido al programador por muy diversos caminos.

Con el uso de los lenguajes de alto nivel, los modelos de programación aproximaron substancialmente al dominio del problema el de su solución. Sin embargo, los primeros de ellos no estimulaban la disciplina en la creación de programas. En la década de los 70's, con la programación estructurada se resolvieron algunos de esos problemas.

LA PROGRAMACION ESTRUCTURADA

Desde una perspectiva amplia, la programación de computadoras no es más que la representación de la descripción de una parte de la realidad. En esta realidad están en juego acciones y datos.

Las técnicas de programación estructurada se fundamentan en el manejo ordenado y global del programa,. Para ello, se construyen estructuras de datos y algoritmos que se interrelacionan y coexisten en función de las necesidades del programador.

En la programación estructurada, los datos pueden ser consultados y modificados por muy diversos agentes y de la misma manera, las acciones comparten información global dentro del programa.

Niklaus Wirth destaca la ineludible relación existente entre la información (EL QUE) y el algoritmo(EL COMO) para llegar a un fin, estos dos elementos se encuentran separados bajo el esquema de la programación estructurada.

Peter Wegner[44] resalta las ventajas de simplicidad, entendibilidad y modificabilidad de los programas estructurados.

Lo anterior se deriva de las características principales de la programación estructurada: *Secuencia* que rige el control por omisión de un programa estructurado, *Selección* que permite al programa optar en una alternativa, dependiendo de alguna condición, e *Iteración* que ejecuta una serie de instrucciones de manera repetitiva.

La programación estructurada sentó las bases de su éxito en dichas características y en el concepto de modularidad: atributo de los programas que consiste en la definición de secciones autocontenidos y mutuamente cohesivos que permiten la construcción de sistemas de manera ordenada y con con un grado importante de depurabilidad y reutilización.

No obstante lo expuesto líneas antes, la construcción y manejo de grandes sistemas, se mantuvo con un alto grado de dificultad. La programción estructuradn no resolvió todos los problemas cuando las soluciones eran muy grandes, complejas y altamente evolutivas, estó se presentó ya que en dichos sistemas los cambios en el problema que resolvían, se presenta en secciones específicas del mismo. El reflejo de los cambios en la solución (compuesta de codigo actuante sobre datos) se disemina por lugares insospechados del sistema creciendo el peligro de no contemplar efectos colaterales cauzados por la modificación.

Con la programación estructurada surgió el concepto de subprograma, inventado antes de 1980, pero en esta ocasión como un mecanismo de abstracción. Grady Booch[5] explica que con lo anterior, tres consecuencias se presentaron:

1- La creación de lenguajes que permitieron el uso de una gran variedad de mecanismos de paso de parámetros.

2- El desarrollo de las teorías referentes a las estructuras de control y a las de ámbito o alcance de las declaraciones.

3- Surgieron los métodos de diseño estructurado que guiaron en la utilización de subprogramas como bloque básico de construcción.

Más tarde, aparecieron los módulos, conjuntos de subprogramas lógicamente relacionados. Cada módulo se compone de subprogramas (parte de la totalidad de las acciones) y de datos, (parte de la totalidad de la información). Los módulos pueden ser construidos por diferentes programadores. Desafortunadamente, la inconsistencia entre las consideraciones de cada equipo de programadores no está necesariamente descartada.

CRISIS DEL SOFTWARE

Es precisamente el crecimiento de la complejidad en los sistemas automatizados lo que trajo como resultado la "Crisis del Software".[51] Gran cantidad de conceptos en el dominio del problema y grandes posibilidades en los diferentes y flexibles lenguajes de programación en el dominio de la solución, establecen el marco en que se presenta dicho fenómeno; para superarlo, se buscaron y estudiaron nuevos niveles de organización.

Según, Mary Shaw[40], investigadora de la Universidad Carnegie Mellon, "El nivel de arquitectura de software requiere nuevos tipos de abstracciones que capturen las propiedades esenciales de los sistemas y de las maneras en que estos interactuan".

TIPO DE DATO ABSTRACTO Y OBJETO

A consecuencia de lo anteriormente expuesto, surge de manera natural el concepto de tipo de dato abstracto (ADT, por sus siglas en inglés), entidad que encapsula los datos y el comportamiento en una sola pieza de código.

Sin el uso del término "Object-Oriented" pero sí con muchas de sus características, el lenguaje Simula en 1967, adopta para sí estos conceptos. A partir de este momento, la tecnología de objetos creció, inició su madurez y se propagó.

Bajo el esquema de la tecnología de objetos, la información y las acciones que la manipulan convergen en un sólo lugar físico y lógico denominado objeto. El comportamiento de esta cápsula se encuentra en función de la clase a la que pertenece.

EL OBJETO Y SUS VENTAJAS

Con el uso de los tipos de datos abstractos y más aún con el de los objetos aumenta la "descriptibilidad" de las porciones de la realidad a modelar, esto en razón de que las funciones del programa coinciden de manera más natural con las acciones del problema.

Adicionalmente, y en opinión de Brad Cox y A.J. Novoblisky,[10] el sistema de objetos que simula la realidad, se encuentra preparado para enfrentar con mayor éxito uno de los más importantes problemas de ésta: El cambio.

Los cambios en un subsistema del mundo real no suelen afectar a datos globales; más bien, alteran trozos de la misma. De esta manera, el trozo

correspondiente del modelo puede ser modificado sin por ello afectar al resto del sistema.

En resumen, se puede decir con el uso del modelo de objetos se cuenta con una mayor probabilidad de tener éxito, al construir soluciones para problemas grandes y complejos así como para modelar sistemas con alto grado de evolutividad.

LENGUAJES ORIENTADOS A OBJETOS

Podemos clasificar a los lenguajes de programación bajo diferentes perspectivas o dentro de un modelo específico. Así tenemos, el modelo imperativo, el declarativo, el orientado a objetos, etc.

La mayoría de los lenguajes populares están circunscritos al modelo imperativo, en realidad la programación orientada a objetos descansa sobre las bases de este modelo, pero dadas sus características ha merecido conceptualizarlo bajo un propio y particular enfoque.

Peter Wegner coloca a Simula 67 como un lenguaje que marca una de las pautas en el desarrollo de los lenguajes de programación. Este lenguaje, sucesor de Algol presenta por primera vez el concepto de clase. Una clase de Simula consiste en un conjunto de declaraciones de datos y de procedimientos seguidos por una secuencia de instrucciones ejecutables. En Simula no obstante la instancia de una clase, no es por completo segura, ya que se permite a través de la notación de punto acceder a los datos internos de la misma. Por otro lado, Simula provee "herencia de clases" , mecanismo de subclasificación muy efectivo para el diseño de jerarquías.

ADA surgió como respuesta a la crisis del software que imperaba en el departamento de defensa de los Estados Unidos. Ada es un lenguaje imperativo de programación para propósitos generales que provee el uso de multitarea, incluye el uso de un tipo de dato abstracto denominado paquete (package). El paquete no es más que un objeto aunque no se maneja el concepto de clase ni mucho menos de herencia.

En la década de los 70's, el grupo de desarrollo de Xerox encabezado por Alan Kay presentó Smalltalk. Este lenguaje se fundamenta en el concepto de clase Un objeto es una instancia de alguna clase, el acceso a los datos de un objeto se da solamente a través de los métodos del mismo. Existe herencia entre clases y cada clase es tambien considerada como un objeto.

Bjarne Straustrup de os laboratorios Bell de AT&T crea C++, lenguaje de programación derivado de C y de Simula 67. C++ provee el uso de objetos como instancias de clases, permite la herencia y aporta el uso de herencia múltiple. Por la naturaleza híbrida de este lenguaje, se mantienen las estructuras tradicionales de C. Se puede permitir el acceso directo a los datos internos de un objeto. Se presentan nuevas categorías bajo las cuales los componentes del objeto se pueden declarar. C++, por otro lado, presenta algunos cambios en relación a la nomenclatura de los conceptos presentados por sus antecesores .

Objective C, creado por Brad Cox se deriva de C++ y de Smalltalk. Este lenguaje es muy semejante a C++ pero sin algunas de sus restricciones.

Eiffel, creado por Bertrand Meyer, es un lenguaje equiparable a Ada pero con características orientadas a objetos semejantes a Smalltalk. Eiffel es considerado como un lenguaje orientado a objetos puro, además presenta algunas ventajas como el manejo de precondiciones y postcondiciones para asegurar el correcto funcionamiento de las acciones.

CONCEPTOS DE PROGRAMACION ORIENTADA A OBJETOS

La programación orientada a objetos es una filosofía de trabajo que se origina en los lenguajes de computación, a partir de estos se propaga y llega al ámbito de las metodologías de análisis y diseño. En este recorrido de 25 años se han originado términos nuevos y adaptado algunos otros para describir los conceptos que dan soporte a esta tecnología o que se generan como consecuencia de los fenómenos de la misma.

En esta sección se presentan los términos y los conceptos que aquellos describen. Lo anterior bajo el contexto general y dentro de la tecnología orientada a objetos.

Se presentan de manera formal las definiciones que los expertos más calificados en el área han publicado con respecto a los conceptos y características de la programación orientada a objetos y una definición a manera de resumen (*en itálicas*) de cada término presentado.

CONCEPTOS DE PROGRAMACION ORIENTADA A OBJETOS

OBJETO: Conjunto de datos y operaciones que pueden acceder a estos datos. BradCox[10]

Colecciones de operaciones y un estado que recuerda el efecto de las operaciones PeterWegner.[15]

Habitantes aislados y únicos de un universo de otra manera vacío Luis Joyanes A.[22]

Conjunto de información y comportamiento que altera dicha información.

CLASE: Orden en que se consideran comprendidas diferentes personas o cosas. Diccionario de la Lengua española.

Conjunto de objetos que comparten una estructura y un comportamiento común. Grady Booch.[5]

Modelo a partir del cual se crean instancias de objetos. Luis Joyanes A.[22]

Unidad básica de modularización que describe una implementación de un tipo de datos abstracto. Bertrand Meyer.[33]

Unidad o modelo en que se consideran comprendidos objetos con características y comportamiento común.

MENSAJE: Operación que un objeto desempeña sobre otro. Grady Booch[5]

Solicitud enviada a un objeto para cambiar un estado o pedir un valor. Ann L. Winblad.[48]

Operación que un objeto desempeña sobre otro para solicitar un valor o cambiar un estado.

METODO: Procedimiento para llevar a cabo un fin. Diccionario de la Lengua española.

Código que aporta funcionalidad a un objeto Brad. J. Cox[10]

Algoritmos que son desempeñados por un objeto en respuesta a la recepción de un mensaje DigiTalk.[7]

Algoritmos desempeñados por un objeto en respuesta a la recepción de un mensaje.

CARACTERÍSTICAS DE LENGUAJES ORIENTADOS A BJETOS

POLIMORFISMO: 'Jn objeto, denotado por su nombre puede responder de diferente manera a un conjunto común de operaciones.

Grady Booch.[5]

Capacidad del mismo mensaje para ser interpretado de forma diferente al ser recibidos por objetos distintos.

Ann Winblad.[48]

Posibilidad de que diversos objetos actúen de modo diferente en respuesta a una misma llamada.

Luis Joyanes A.[22]

Fenómeno que consiste en la posibilidad de que un mensaje se vea respondido de distintas maneras dependiendo del objeto que lo recibe.

ABSTRACCION: Consideración aislada de las cualidades de un objeto.

Diccionario de la lengua española

Características esenciales de un objeto que lo distinguen de todos los demás.

Grady Booch.[5]

Proceso de crear una superclase extrayendo cualidades comunes y características generales a partir de clases u objetos más específicos.

Ann Winblad.[48]

Consideración de las características esenciales de un objeto sin atender al resto de las mismas.

HERENCIA: Relación entre clases donde una comparte la estructura y conducta de una o más de otras clases.

Grady Booch.[5]

Técnicas para definir nuevos tipos de datos describiendo los aspectos en los que cada tipo difiere de algún otro tipo que existiera con anterioridad.

Brad J.Cox.[10]

Relación entre clases que permite que las instancias u objetos de una, adquieran las propiedades de las instancias u objetos de otra.

ENCAPSULACION: Conjunto de operaciones con una interface visible y con ocultamiento de la realización del objeto.

Ann Winblad.[48]

Proceso de ocultamiento de todos los detalles de un objeto que no contribuyen a sus características esenciales.

Grady Booch.[5]

Protección de los datos privados de un objeto contra el acceso externo.

Brad. J. Cox[10]

Propiedad de los objetos que consiste en ocultar los datos del objeto y solo permitir acceso a través de llamadas a los procedimientos internos.

MODULARIDAD: Propiedad de un sistema que ha sido descompuesto en módulos cohesivos y separados.

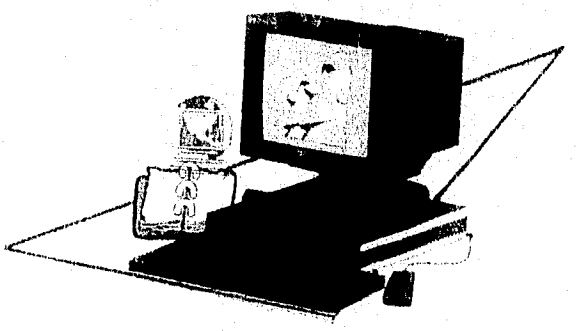
Grady Booch.[5]

Módulo de computación de universo abierto en el cual la relación entre las partes individuales y el todo no necesita ser conocido y declarada en el momento de compilación.

Brad. J. Cox.[10]

Propiedad de un sistema que se ha descompuesto en secciones autocontenidas y cuya relación con el resto del sistema no es esencial.

III ENSEÑANZA DE LA PROGRAMACION ORIENTADA A OBJETOS



III LA ENSEÑANZA DE LA PROGRAMACION ORIENTADA A OBJETOS

Durante la década de los 70's se presentan las bases conceptuales que dan soporte a la tecnología orientada a objetos. A partir de mediados de los 80's se presta especial atención al proceso de enseñanza-aprendizaje de éste conjunto de técnicas.

La literatura publicada muestra diferentes enfoques para abordar estos temas; no obstante, podemos identificar algunos puntos generales de coincidencia:

-Curriculum para la enseñanza del paradigma orientado a objetos
-Apoyos didácticos para la enseñanza del paradigma orientado a objetos

CURRICULUM PARA LA ENSEÑANZA DEL PARADIGMA ORIENTADO A OBJETOS

Para estudiar el curriculum (programa o plan de estudios) para la enseñanza de la tecnología orientada a objetos se debe abordar el siguiente conjunto de conceptos:

-Enfoques de enseñanza
-Objetivos del curriculum
-Cualidades de un buen curriculum
-Secuencia de exposición
-Estrategias de impartición del curriculum
-Obstáculos para la impartición del curriculum
-Costo de enseñanza

Enfoques de enseñanza

Para realizar un buen programa de enseñanza de tecnología orientada a objetos(TOO) se debe iniciar por la identificación de perfil de la persona o grupo a quien va dirigida la instrucción. Desmod D'Souza[11] marca la diferencia entre **educación** y **entrenamiento** explicando que el entrenamiento consiste en la impartición de lo que se requiere en un momento dado, mientras que la educación sienta las bases conceptuales a la vez que prepara con mayor formalidad.

Podemos, de lo anterior deducir que existen dos enfoques generales en la exposición de la TOO:

-Académico y
-Empresarial

El enfoque académico descansa sobre bases educativas, tiene fines formativos básicamente e informativos, da sustento al conocimiento del estudiante y lo prepara en un plano teórico y práctico. Un alumno que recibe formación profesional es capaz de aplicar los conocimientos adquiridos acerca de los paradigmas de programación y desarrolla soluciones en el lenguaje que sea adecuado o del que se tenga disponibilidad, y es importante resaltar que el estudiante universitario se prepara, aprende y adquiere la capacidad de investigar y aprender por cuenta propia.

Por otro lado, el enfoque empresarial tiene sus bases en un esquema de entrenamiento, de actividades realizadas por mentores que siguen de cerca el desarrollo de proyectos productivos y finalmente por consultores que dan soporte relativamente permanente al grupo que recibió la instrucción.

La tecnología orientada a objetos por su misma naturaleza (relativamente joven, util y exitosa), es objeto de aprendizaje tanto en las universidades como en las empresas; un análisis de la manera en que se enseña y los posibles métodos y herramientas para facilitar este proceso, incluye ambos enfoques.

Objetivos del curriculum

Kent Beck[3] presenta en su conferencia "A Laboratory for teaching Object-Oriented Thinking" durante la OOPSLA 1989:

> **"El problema más difícil de la enseñanza de la programación orientada a objetos es transmitir al estudiante el conocimiento global del control que es posible tener con programas procedimentales y distinguir el conocimiento local de los objetos para cumplir con sus tareas... ...enseñar objetos se reduce a enseñar diseño orientado a objetos"[3]**

Por otro lado, D'Souza señala:

> **" El objetivo que debe tener un curriculum de OO es exponer nuevas técnicas y conceptos a un nivel suficiente para impartir una apreciación de la aplicabilidad de la tecnología y de su relación costo-beneficio, desde el nivel de paradigma hasta el nivel de diseño selectivo y el de implementación" [12]**

C. Thomas Wu[53] establece los siguientes objetivos más particulares :

-Impartición de lenguajes OO
-Enseñanza de diferentes metodologías de análisis y diseño de OO
-Presentación de conceptos avanzados de programación OO

Finalmente Tsvi Bar- David en su artículo "Object Oriented Education and Training in the 1990's"[42] marca una serie de objetivos desde un punto de vista mas general :

-Reforzar la cultura de la reutilización tanto mental como de software
-Integración de análisis orientado a objetos (AOO), diseño orientado a objetos (DOO) e Implementación,
-Integración a través de especificaciones formales de sistemas OO

Con base en lo anterior se puede decir que enseñar tecnología OO, tiene que ver no tanto con el aprendizaje de un nuevo lenguaje sino con una manera diferente de pensar.

Pueden establecerse los siguientes objetivos que deben ser cumplidos por un buen curriculum de tecnología orientada a objetos como propuesta de este trabajo recepcional.

Objetivo General: Fomentar la cultura de la reutilización y enseñar a pensar en términos de entidades autocontenidas y cohesivas.

Objetivos Particulares :

1.-Integración de análisis, diseño e implantación orientado a objetos
2.-Aprendizaje de diferentes metodologías de análisis y diseño OO
3.-Aprendizaje de diferentes lenguajes OO

Cualidades de un buen curriculum

Janet Conway[9] de General Electric Advanced Concepts Center lista una serie de requerimientos nuevos de software:

-Reusabilidad de código
-Simplicidad de diseño
-Interfaces de usuario bien definidas
-Compatibilidad con código existente
-Portabilidad
-Autonomía de datos
-Extensibilidad de código

C. Thomas Wu[33] establece los siguientes objetivos más particulares :

-Impartición de lenguajes OO
-Enseñanza de diferentes metodologías de análisis y diseño de OO
-Presentación de conceptos avanzados de programación OO

Finalmente Tsvi Bar- David en su artículo "Object Oriented Education and Training in the 1990`s"[42] marca una serie de objetivos desde un punto de vista mas general :

-Reforzar la cultura de la reutilización tanto mental como de software
-Integración de análisis orientado a objetos (AOO), diseño orientado a objetos (DOO) e Implementación,
-Integración a través de especificaciones formales de sistemas OO

Con base en lo anterior se puede decir que enseñar tecnología OO, tiene que ver no tanto con el aprendizaje de un nuevo lenguaje sino con una manera diferente de pensar.

Pueden establecerse los siguientes objetivos que deben ser cumplidos por un buen curriculum de tecnología orientada a objetos como propuesta de este trabajo recepcional.

Objetivo General: Fomentar la cultura de la reutilización y enseñar a pensar en términos de entidades autocontenidas y cohesivas.

Objetivos Particulares :

1.-Integración de análisis, diseño e implantación orientada a objetos
2.-Aprendizaje de diferentes metodologías de análisis y diseño OO
3.-Aprendizaje de diferentes lenguajes OO

Cualidades de un buen curriculum

Janet Conway[9] de General Electric Advanced Concepts Center lista una serie de requerimientos nuevos de software:

-Reusabilidad de código
-Simplicidad de diseño
-Interfaces de usuario bien definidas
-Compatibilidad con código existente
-Portabilidad
-Autonomía de datos
-Extensibilidad de código

Con la intención de preparar creadores de software que cubran con estos requerimientos y que cumplan con los objetivos marcados previamente, se debe establecer un conjunto de cualidades con las cuales un buen curriculum de TOO debe contar.

James C. McKim Jr.[32] de Hartford Graduate Center diseña sus cursos de TOO fijando 3 grandes objetivos cualitativos:

- Los estudiantes deben aprender a usar los conceptos OO
- los estudiantes deberán tener la oportunidad de aprender por si mismos si el modelo tiene las cualidades que se dice que tiene.
- El curso debe ser divertido.

Por otra parte, D'Souza[16] de Icon Computing sugiere las siguientes cualidades con las que debe contar la educación OO:

-Clara
-Consistente
-Que presente los fundamentos
-Ordenada
-Que represente un reto
-Relevante (aplicable).

De lo anterior se desprende la siguiente propuesta de este trabajo recepcional que un buen curriculum OO debe ser:

1.- Completo
2.- Claro
3.- Consistente y ordenado
4.- Teórico y práctico
5.- Con posibilidad para el estudiante de demostrar por si mismo lo aprendido.
6.- Aplicable
7.- Divertido

1.-Completo: El curriculum debe cumplir tanto los fundamentos como los conceptos avanzados de TOO, así como con todas las fases de las que se compone un proyecto.

2.-Claro: La totalidad del curriculum debe presentarse de forma clara dedicando más tiempo a la parte inicial para que los fundamentos lleguen a ser una "Segunda naturaleza".
Para una mayor claridad debe utilizarse una notación simple y probada.

3.- Consistente y ordenado: El programa de enseñanza debe seguir una secuencia lógica que posea continuidad a lo largo de los diferentes temas.

4.- Teórico y práctico: El programa debe contemplar tanto la enseñanza de los conceptos, teóricos acentuados en la primera etapa como las sesiones prácticas y desarrollo de proyectos donde se aplique lo aprendido.

5.- Con posibilidad para el estudiante de demostrar por sí mismo lo aprendido: Esto se da con la práctica, con ejemplos que se apliquen a lo largo de cada etapa del curso y con herramientas interactivas efectivas.

6.- Aplicable: El programa deberá estar estructurado de tal manera que el egresado logre aplicar los conocimientos al desarrollo de sistemas reales.

7.-Divertido: "Este criterio no se aplica a otras ciencias salvo en computación....." ".....trabajan más eficientemente los estudiantes que más se divierten".[32]
Para que el curso tenga esta característica se sugiere el desarrollo de un proyecto total que según James C. McKim[32] da mejores resultados si es un sistema lúdico.

Secuencia de exposición

Los temas que debe incluir un buen curriculum OO así como el orden en que estos se presentan es determinante para que cumpla con las cualidades (1),(3), y (4)

Durante la OOPSLA '1989', Karl J. Lieberherr y Arthur J. Riel[7] de Northeastern University presentan en su "Contribution To Teaching Object Oriented Design and Programming" La primera secuencia de curriculum publicado. Los autores describen su esquema en un orden de complejidad ascendente e incluyen detalles de implanacion (utilizando lenguaje de OO) al final de cada una de cuatro etapas (ver tabla 3.1)

Fase 1	Fase 2	Fase 3	Fase 4
CLASE		HERENCIA	CLASES
OBJETO	HERENCIA	MULTIPLE	PARAMET
METODO			MODULOS
IMPLEMEN	IMPLEMEN	IMPLEMEN	IMPLEMEN

tabla 3.1 Secuencia de curriculum segun Lieberherr[27]

Como se aprecia aún no se incluyen las etapas generales de la vida de un sistema orientado a objetos

Es hasta febrero de 1991 cuando Elizabeth Gibson[26] publica una propuesta más completa para un curriculum OO orientado a programadores (ver tabla 3.2) a analistas y diseñadores (ver tabla 3.3) y a gerentes (ver tabla 3.4)

Tema y Secuencia	Formato educacional
Introducción a un lenguaje OO	Lecturas y ejercicios
Programación de un mes	Trabajo independiente
Metodología: Análisis del Comportamiento de los objetos. Diseño orientado a objetos	Lecturas y ejercicios
Prog. de nivel medio	Lecturas y ejercicios
Desarrollo de aplicaciones	Trabajo independiente Revisiones de diseño Consultoría general
Prototipos y producto	Lecturas Trabajo independiente Revisiones de diseño Consultoría general

Tabla 3.2 Secuencia educacional recomendada para programadores OO[26]

Tema y Secuencia	Formato educacional
Introducción a tecnología OO términos e impacto	Seminario:Lecturas, ejercicios ilustrativos
Metodología: Análisis del Comportamiento de los objetos. Diseño orientado a objetos	Lecturas y ejercicios
Manejo de proyectos OO	Lecturas y ejercicios
Selección de primeras aplicaciones para desarrollo	Trabajo independiente Consultoría general
Desarrollo de destrezas de análisis y diseño	Trabajo independiente Consultoría general

Tabla 3.3 Secuencia educacional recomendada para analistas y diseñadores[30]

Tema y Secuencia	Formato educacional
Introducción a tecnología OO términos e impacto	Seminario:Lecturas, ejercicios ilustrativos
Análisis del Comportamiento de los objetos	Lecturas y ejercicios

Tabla 3.4 Secuencia educacional recomendada para gerentes[30]

Mas tarde y bajo un enfoque más academico Richard Wiener[31] presenta en 1993 la siguiente secuencia

1 - Características esenciales
Teoría de clasificación
herencia y composicion
mecanismos de creación de objetos
envio de mensajes
2 - Ilustracion de las propiedades y caracteristicas de la solucion de problemas a través de un lenguaje OO puro
3 - Análisis OO, Diseño OO y Validacion
4 - Integración de los tres primeros pasos para la solución de un problema.

Wiener incluye herramientas Eiffel y SmallTalk en sus cursos

El tema de la secuencia es abordado más directamente por James C. McKim[32] al presentar el siguiente orden:

1.- Conceptos Generales
2.- Encapsulación
3.- Herencia
4.- Polimorfismo
5.- Lenguaje OO

Para su última fase este autor utiliza Eiffel .

Bertrand Meyer[33] Generaliza de la siguiente forma :

1.-Estructuras de datos y algoritmos OO
2.-Ingeniería de Software OO
3.-Análisis y diseño OO

Para impartir un curriculum bajo cualquier secuencia, los autores proponen diversas técnicas, notaciones y herramientas; no obstante, se puede proponer a partir de lo expuesto, la siguiente secuencia teórica:

1.-Introducción y antecedentes
2.-Conceptos generales
3.-Lenguaje Orientado a objetos puro
4.-Ingeniería de Software orientada a objetos
5.-Aplicación práctica Integradora

1.- Introducción y antecedentes

Todas las publicaciones disponibles desde 1990 presentan una fase de introducción que consiste en una justificación del aprendizaje de TOO. Vale la pena agregar antecedentes históricos relacionados con:

Ciencias de la computación
Ingeniería de Software y
Lenguajes de programación.

2.- Conceptos generales

Los conceptos y características estudiadas en esta sección son:

PARTE 1	PARTE 2
Objeto	Encapsulación
Clase	Herencia simple y múltiple
Método	Polimorfismo y ligadura dinámica
Mensaje	Genericidad y clases parametrizadas

3.- Lenguaje Orientado a objetos puro

Chamond Liu y su equipo[28] durante la OOPSLA 1992 presentan las conclusiones de su experiencia obtenida impartiendo cursos de TOO con Smalltalk por un lado y C++ por el otro. Los autores reportan el aumento de tiempo en la impartición del curso apoyando en C++ como vehículo de aprendizaje por la dificultad lingüística que este representa.

El efecto anterior es compatible con los resultados de Wiener[47], pero este autor recomienda la enseñanza de un lenguaje OO híbrido como C++, Objective C, CLOS u otro pero solo después que se ha logrado el aprendizaje de un lenguaje OO puro como Eiffel o Smalltalk (este último impartido después de los conceptos fundamentales).

James C. McKim, establece:

"Muchos declaran que C++ es el mejor lenguaje disponible para escribir software real pero pocos lo recomiendan para la enseñanza, su compatibilidad hacia atrás con C crea muchas distracciones" [32]

Este autor elige Eiffel sobre Smalltalk por la verificación estática de tipos y la herencia múltiple.

La decisión aquí es la siguiente: Si se cuenta con poco tiempo se debe elegir Smalltalk o Eiffel, éste último si se tiene acceso a recursos no limitados, en cambio si el tiempo no es determinante se puede incluir C++ pero como segundo lenguaje OO.

4.- Ingeniería de Software orientada a objetos

Dentro de este punto se incluye análisis, diseño e implantación OO, utilizando como sugiere Wiener[47] los conceptos expresados por Meyer[33] y Booch[5] con la metodología y notación definida en Rumbaugh. Además se pueden incluir los conceptos de desarrollo multipersonal de software, técnicas de manejo de proyectos, economía, métrica, etcétera, presentados por Meyer.[33]

5.- Aplicación práctica Integradora

En realidad esta fase debe iniciar en paralelo al curso y proseguir conforme avanza el curriculum, su objetivo es ampliar los conocimientos adquiridos y permite que el programa cuente con las cualidades (4, 5 y 6) y si el trabajo resulta suficientemente interesante se puede cumplir con la cualidad 7 (curso divertido).

Estrategias de impartición del curriculum

Las diferentes propuestas de curriculum publicadas en la literatura disponible presentan algunos lineamientos generales definidos para apoyar el diseño del curriculum y dar soporte al plan de estudios en su totalidad. Elizabeth Gibson[20] incluye en su estrategia de diseño de un buen curriculum lo siguiente:

-Extraer la experiencia de expertos desarrolladores OO
-Empaquetar dicha experiencia en un curriculum que pueda ser asimilada por neófitos de esa tecnologia.
-Ampliar y ajustar el curriculum para cumplir con las necesidades de diversas audiencias.
-Desarrollar medios de transferencias de tecnología hacia grandes organizaciones con diferentes culturas.

Dentro de este orden de ideas, durante el panel "Managing theTransition to Object-Oriented Technology"[26] en la OOPSLA 1991, sus participantes coinciden en que una base importante del entrenamiento en TOO es el desarrollo incremental de prototipos con estrategias de aprendizaje por :

Conceptualización
Ejemplo y analogía
Experiencia
Ejercicios de enriquecimiento

Los enfoques estratégicos de enseñanza "Bottom-Up" y "Top-Down" son presentados por C. Thomas Wu.[53]

Dentro de la estrategia "Top-Down" se inicia con los conceptos hasta llegar a un lenguaje específico, en tanto que con "Bottom-Up" se inicia con el estudio de un lenguaje y se analizan los conceptos aplicados a él .

Wu puntualiza: **"El enfoque Top-Down es probablemente más benéfico para programadores avanzados y para principiantes puede ser ambiguo y nebuloso. El enfoque Bottom-Up es más adecuado para los principiantes, el problema aquí es que ellos pueden fácilmente igualar programación OO con el lenguaje que ellos aprenden primero"**

Thomas Wu propone una estrategia Bottom-Up en cuatro fases :

0.-Introducción (Contextualización)
1.-"Non-OOP" (Programación tradicional)
2.-"Semi-OOP" (Programación como cliente)
3.-"Full-OOP" (Programación cliente-servidor)

Por otro lado Bertrand Meyer[33] sugiere el uso del término "Caja negra" refiriéndose a los objetos de alguna clase a quienes se les envía mensajes y responden de alguna manera esperada y propone la estrategia de "apertura progresiva de las cajas negras" así como la aplicación de "curriculum invertido".

Se puede concluir que las estrategias y lineamientos generales se fijan en función de los siguientes factores para con base en ellos enfocar la enseñanza:

-Perfil y objetivos de la audiencia
-Herramientas computacionales disponibles

Obstáculos para la impartición del curriculum

Como cualquier programa educativo las barreras que pueden dificultar el proceso de enseñanza dependen de la actitud de la audiencia, pero con base en un estudio de la empresa IBM[28] se concluye lo siguiente: El estudiante muestra menor receptividad y rendimiento cuando su formación anterior tiene las siguientes características:

-Es reducida la cantidad de líneas de código escritas en su vida
-Cuando ha pasado un tiempo considerable de su más reciente experiencia de programación.
-Cuando no conoce ningún lenguaje de programación
-Cuando conoce menos de cuatro familias de lenguajes (el efecto Dogma)
-Cuando el único lenguaje que conoce es Basic.

Por el lado opuesto y considerando el desarrollo del curso se ha encontrado mejor desempeño en los estudiantes que :

-Invierten más horas explorando cuestiones más allá de lo aprendido en los laboratorios
-Disfrutan la programación
-Cuando aprende lenguajes fuera de su trabajo o estudio formal

David F Himant[21] destaca que en una empresa la capacitación en OO es importante para ambos: técnicos y gerentes y si el tiempo y el presupuesto lo permiten, se debe ofrecer entrenamiento a miembros de otros departamentos, ya que estos pueden ser nuestros aliados para minimizar obstáculos y llegar al éxito.

Costo de enseñanza

Desmond D'Souza en su artículo " The Cost of Object Education" concluye:

" El programa de entrenamiento y educación es uno de los más caros componentes al hacer la transición hacia la tecnología OO"[18]

D'Souza propone los siguientes rubros para ser incluidos en el costo de dicha transición:

-El costo del tiempo de los estudiantes por la duración del curso
-El costo del curso mismo (con variación de calidad y resultados)
-El costo de viaje si el curso no es en la localidad
-El costo de retardo en desarrollo de proyectos
-El costo de administración (Identificación, negociación y elección de servicios)

y estima un rango de costo entre 28,000 y 66,000 dólares para un grupo de 10 a 20 estudiantes a capacitar en un curso con duración de 5 dias.

Apoyos didácticos para la enseñanza de la tecnología orientada a objetos

Existen algunos apoyos didácticos para facilitar cualquier proceso enseñanza-aprendizaje. La impartición de un curriculum OO se auxilia por apoyos tradicionales como los son:

Lecturas
Ejercicios
Proyección de diapositivas y acetatos, etc

Amado a lo anterior, un instructor puede valerse de herramientas alternas.

Una herramienta interesante que apoya una técnica hoy ya muy conocida son las tarjetas de responsabilidades y colaboradores ("CRC cards")[48]. Esta herramienta y técnica consiste en la utilización de tarjetas pequeñas de cartón. En cada una de ellas se escriben los datos de una clase: sus responsabilidades (metodos) y las clases colaboradoras. Las tarjetas CRC se constituyen tanto como apoyo al análisis y diseño OO como a la enseñanza.

Existen además apoyos programados interactivos, es el caso de los ambientes para producción como Eiffel, Smalltalk o KEE[25] y sistemas educativos "per se" como Demeter.

Renate Kemp En OOPSLA 1987[25] presenta el sistema interactivo KEE de Intellicorp y establece que un ambiente de programación que colabore a la enseñanza debe tener las siguientes características:

-Logre que el alumno interactúe con objetos, fácil acceso a los objetos, a las relaciones entre ellos, a sus atributos y a sus conductas.

-Un alto grado de interactividad que haga inmediatamente visibles los cambios en los objetos.

Thomas Wu establece:

"La gente aprende mejor por medio de experimentos nuevos... ...Para hacer que los conceptos realmente permanezcan en las mentes de los principiantes necesitamos herramientas de programación que ilustren, resalten , reiteren y ejemplifiquen en forma concreta con un valor pedagógico." [53]

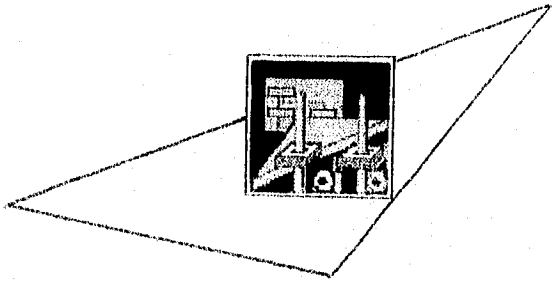
Wu propone además las cualidades que debe tener una herramienta con dicho objetivo:

Fácil de aprender
Poderoso
Compatible con lenguajes de programación orientada a objetos
Capaz de ilustrar alternativas
Exploratorio.

Por lo anterior se puede decir que además de los propios intérpretes, compiladores y herramientas CASE ("Computer Aided Software Engineering) que sirven para ilustrar lo aprendido a través de su uso normal se recomienda ampliamente el uso de herramientas interactivas que refuercen la enseñanza y que posean las cualidades previamente citadas las cuales pueden ser complementadas y presentarse de la siguiente lista ordenada bajo un criterio de importancia descendente:

Compatible con conceptos de orientación a objetos
Compatible con algún(os) lenguaje(s) orientados a objetos
Llamativo
Interactivo
Poderoso
Exploratorio

IV LABORATORIO GRAFICO BASICO DE PROGRAMACION ORIENTADA A OBJETOS



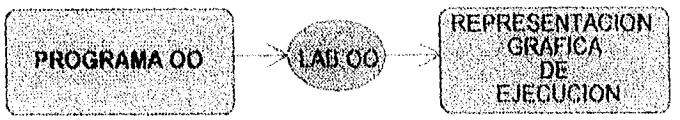
IV LABORATORIO GRAFICO BASICO DE PROG.ORIENTADA A OBJETOS

OBJETIVOS

El laboratorio grafico basico de programacion orientada a objetos (Lab OO)es un sistema de *software* cuyo objetivo es facilitar al estudiante de programacion, el entendimiento de los conceptos que fundamentan la tecnologia orientada a objetos.

MEDIOS (Especificacion general del sistema)

Se propone el uso de un sistema interactivo que reciba como entrada un programa escrito en un lenguaje orientado a objetos y produzca como salida una representación gráfica que facilite el entendimiento de lo que el programa de entrada realizaria conceptualmente al ser ejecutado.



DIRIGIDO A

Profesores, capacitadores y mentores en tecnologia orientada a objetos para que a su vez lo utilicen en la enseñanza de esta disciplina a sus estudiantes.

A los autodidactas de programación orientada a objetos.

ALCANCES

Se busca facilitar el entendimiento de los conceptos que dan soporte a la programación orientada a objetos lo cual permite facilitar la transición en la forma de pensar en dirección hacia la tecnologia OO.

Los conceptos que se busca clarificar son:

Encapsulación,
Herencia y
Polimorfismo.

LIMITACIONES

No se busca facilitar el aprendizaje o entrenamiento de las metodologías de análisis y diseño OO sino más bien el reforzar el entendimiento de los mecanismos y abstracciones que tienen lugar al construir un programa orientado a objetos.

No se busca el facilitar el aprendizaje de un lenguaje de programación en especial.

Podemos decir que, derivado del objetivo y de la especificación general del sistema surgen los siguientes cuestionamientos:

1.- ¿En qué lenguaje deberá ser escrito el programa OO que servirá como entrada al Lab OO?

2.- ¿Qué subconjunto del lenguaje elegido será considerado?

3.- ¿En qué consiste la representación gráfica que mostrará el Lab OO? y

4.- ¿En qué lenguaje se desarrollará el sistema Lab OO?

La respuesta a estas cuatro preguntas describirá el sistema, su capacidad y sus limitaciones prácticas.

LENGUAJE OO PARA EL PROGRAMA DE ENTRADA AL SISTEMA

Como se planteó en el capítulo anterior, existen en la actualidad muchos lenguajes considerados como orientados a objetos, entre ellos destacan:

SIMULA
SMALLTALK
C++
EIFFEL
OBJECTIVE C

Cada uno de ellos posee características que los distinguen de los demás la elección del lenguaje apropiado debería considerar las cualidades y defectos que dichos lenguajes presentan y ponderar los criterios con base en lo que nos interesa destacar en el proceso de enseñanza-aprendizaje de la POO.

Debemos tomar en cuenta que lo que se busca es una herramienta que auxilie en la enseñanza, por tanto el lenguaje debe contar con implantaciones para plataformas a las que el mayor número de estudiantes tenga acceso. Además del acceso simple, el lenguaje debe tener un grado importante de aceptación para que la enseñanza del mismo se presente. Finalmente y atendiendo a los objetivos planteados, es importante que el lenguaje tenga un alto grado de coincidencia con los conceptos puros de la programación orientada a objetos [28]

Por lo anterior, podemos establecer nuestros criterios de decisión como sigue:

PORTABILIDAD
DISPONIBILIDAD (Popularidad)
GRADO DE COINCIDENCIA CON CONCEPTOS OO

Con base en estos criterios podemos ordenar nuestros lenguajes de la siguiente forma:

PORTABILIDAD	PLATAFORMA		
LENGUAJE	DOS	WINDOWS	UNIX
SIMULA	*SI*	-	-
SMALLTALK	*SI*	*SI*	*SI*
C++	*SI*	*SI*	*SI*
EIFFEL	-	-	*SI*
OBJECTIVE C	-	-	*SI*
ADA	*SI*	-	*SI*

Tabla 4.1 Portabilidad de lenguajes basados u orientados a objetos

Por lo anterior escrito, se puede afirmar que tanto Objective C como Eiffel son lenguajes de poca disponibilidad en un contexto general, la elección se plantea entre C++ y Smalltalk. Dada la comparación planteada por Voss[4] y

con base en los conceptos vertidos en OOPSLA 1992[25] 7.5 la elección es el lenguaje Smalltalk.

En lo particular, la propuesta de este trabajo recepcional es la de apegarse en la medida de lo posible a los conceptos reconocidos como puros de la tecnología orientada a objetos, de los lenguajes estudiados se concluye que Smalltalk es el mas adecuado ya que en este lenguaje todas las funciones deben ser colocadas en una clase mientras que en otros lenguajes se permite la existencia de funciones totalmente independientes de clase alguna.

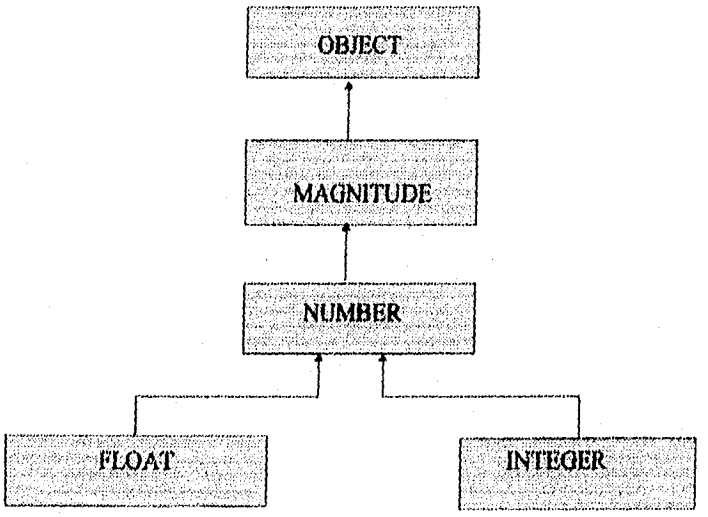
SUBCONJUNTO A SER CONSIDERADO DEL LENGUAJE ELEGIDO

El sistema no pretende ser exhaustivo y dado que sus objetivos contemplan la ilustración de los conceptos de herencia, encapsulación y polimorfismo, se considera tan sólo un subconjunto del árbol de clases total de Smalltalk.

Dado que la descripción de los medios nos indica que el sistema será utilizado con una computadora de uso generalizado, entonces deberá elegirse un interprete de Smalltalk utilizable en este tipo de máquina.

Se considera Smalltalk V de Digitalk.

El subárbol de clases que nuestro sistema contendrá es el siguiente:



REPRESENTACION GRAFICA OO

Se propone una representación gráfica de lo que conceptualmente suceda cuando se ejecuta un programa orientado a objetos. Es una representación muy particular que consiste en lo siguiente:

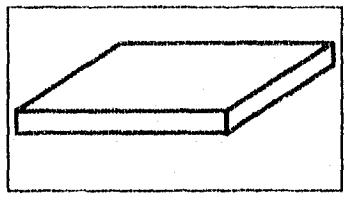
LINEA DE PROGRAMACION

Una línea de PROGRAMACION en un lenguaje orientado a objetos debe interpretarse como sigue.

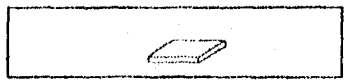
- Existe un objeto receptor, existe un mensaje dirigido al objeto receptor y existen potencialmente otros objetos que sirven como argumentos a dicho mensaje.

REPRESENTACION

-Existe un mundo único de objetos que puede contener hasta m x n objetos simultáneamente.
En el mundo de los objetos, nacen objetos, envían y/o reciben mensajes, responden y o reciben respuestas de otros objetos. finalmente desaparecen los objetos.

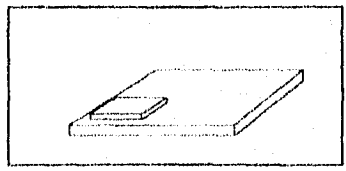


-Un objeto es un habitante del mundo de los objetos.

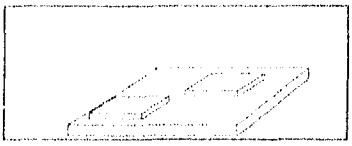


-La creación de un objeto se presenta por su surgimiento en el mundo de los objetos.

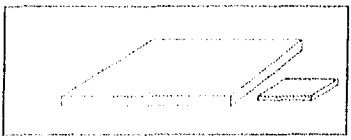
-La destrucción de un objeto se presenta de manera inversa a la de su creación.



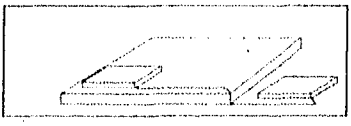
-Un objeto espontaneo es aquel que se crea en el mundo de los objetos por si sólo como consecuencia de ser referido en la línea de programación como objeto receptor o como objeto argumento del mensaje.



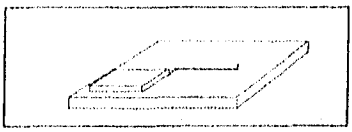
-Existe un ente externo al sistema que es quien escribe el programa, este programador es representado por un Objeto externo, objeto fuera del mundo de los objetos.



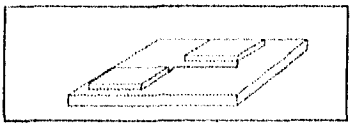
-El mensaje se representa por una línea que sale del objeto emisor y que concluye en el objeto receptor.



-Existe una línea de creación que surge de un objeto que ha recibido un mensaje y como respuesta crea un nuevo objeto.



-Un objeto respuesta es un objeto con las mismas características de los demás objetos pero éste es creado por otro objeto como respuesta de éste último a un mensaje recibido.



LABORATORIO GRAFICO BASICO DE PROGRAMACION ORIENTADA A OBJETOS (LABOO)

El sistema se desarrolló en C++ ya que se requería generar un programa ejecutable que no precisara de un ambiente de ejecución ya que esto aumentaría el costo y reduciría su aplicabilidad.

Las posibilidades de utilización de ambiente gráfico se presentaron con el compilador Borland C++ en su versión 3.1 el cual superó las características de Smalltalk V de Digitalk version 2.0 (que se encontraba disponible) ya que este último utiliza una resolución muy baja en sus gráficos y carece de color.

Con lo anterior y considerando que buena parte del éxito de un sistema de apoyo didáctico depende de su interface de usuario se opto por eliminar el interprete de Smalltalk disponible.

Eiffel no fué considerado ya que existe poca disponibilidad de esta herramienta.

Diseño General del sistema

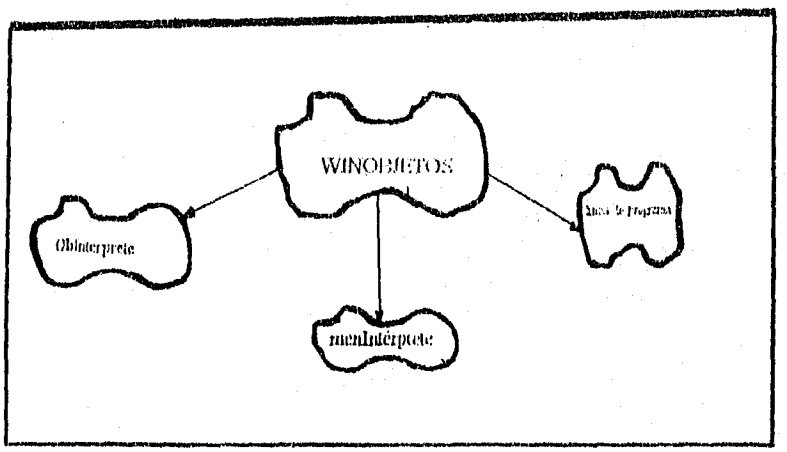


Diagrama general de Clases

Diseño General del sistema

El sistema se compone de las siguientes clases principales en relación a su importancia para la aplicación misma

Línea de Programa,
ObjInterprete y
MenInterprete

La siguiente figura muestra un diagrama de clases según la metodología de Booch en donde se ilustran los mensajes enviados entre las instancias de todas las clases de la aplicación

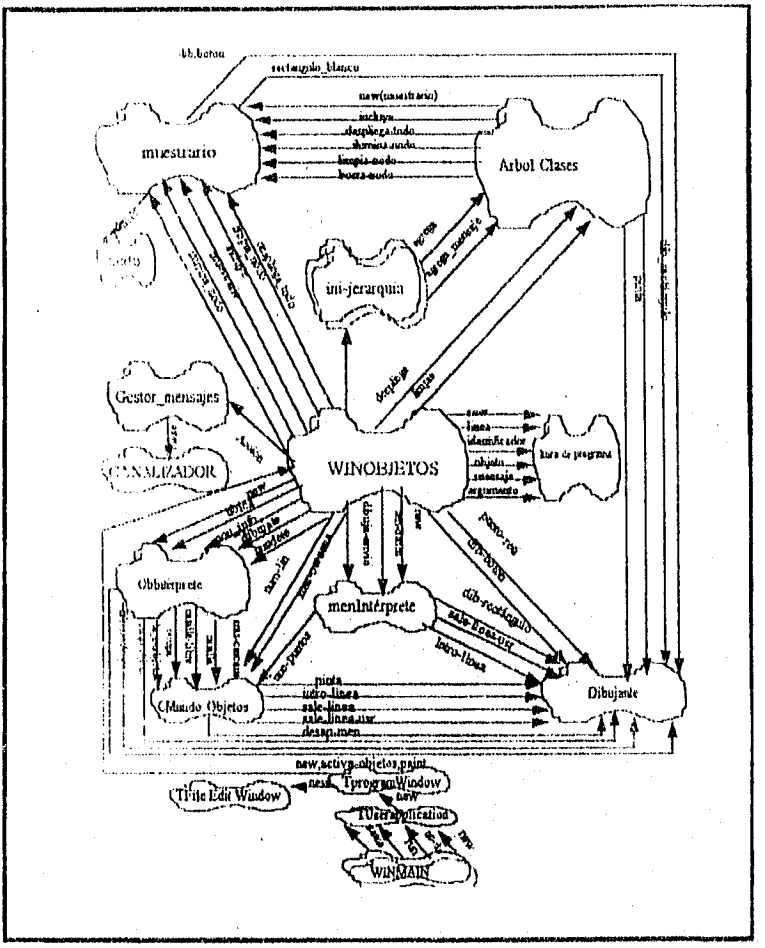


Diagrama general de Clases y mensajes entre las clases

Como se observa en la figura, existe también una clase **WinObjetos** cuya instancia controla el más alto nivel de abstracción y una clase **WinEditor** (TFileEditWindows de Object Windows Library).

Cabe destacar la existencia de **Dibujante**, una clase servidora que se encarga de los trazos de mas bajo nivel de la aplicación.

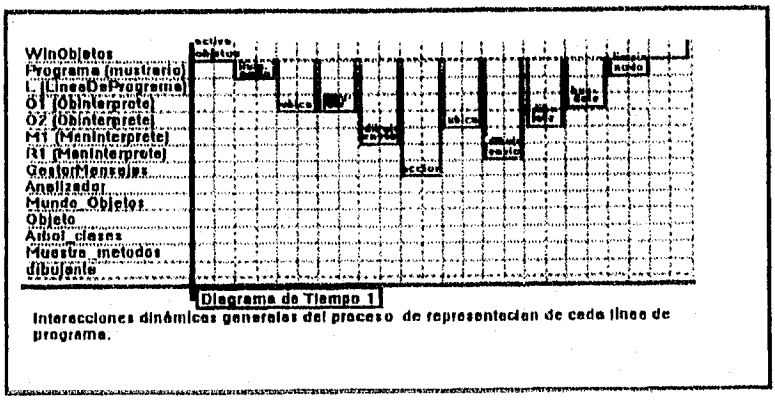
A continuación, se ofrece un panorama general de los procesos que se llevan a cabo en **LabOO** se ilustran con los diegramas de tiempo correspondientes

Proceso general:

El usuario escribe un programa utilizando el editor del sistema. La clase editor cuenta con un método para analizar el texto escrito, verificar su gramática y en caso de éxito envía dicho programa a un arreglo de cadenas.

Una instancia de la clase **WinObjetos** (ventana de objetos) controla el proceso general de análisis de cada instrucción del programa y su correspondiente representación didáctica.

El Método de **WinObjetos** que se encarga de lo anterior se llama **activaObjetos**, la secuencia general del sistema al momento de representar el desarrollo de un programa en lenguaje OO lo desarrolla éste método y la figura siguiente muestra el diagrama de tiempo general del mismo.



Interacciones dinámicas generales del proceso de representacion de cada linea de programa.

activaObjetos en general efectúa lo siguiente:

a).- Crea un objeto de la clase **LíneaDePrograma** el cual recibe el arreglo de cadenas que almacena el programa.

b).- Se solicita al objeto de la clase **LíneaDePrograma** que analice la línea para obtener el Objeto, el Mensaje, el posible Argumento y el posible identificador (Variable).

c).- Crea un objeto de la clase **ObInterprete** con la información objeto del paso anterior y le envía el mensaje **dibújate** para su representación.

d).- Crea un objeto de la clase **MenInterprete** con la información mensaje del paso (b) y le envía el mensaje **dibujaEnvio** para su representación.

e).-Se le envía la información acerca del Objeto, el Mensaje y el posible Argumento a una instancia de la clase **GestorMensajes** la cual realiza todo lo necesario para regresar un objeto respuesta, si es que el mensaje y el objeto receptor son compatibles (Además considera herencia).

f).-De tener éxito el paso anterior, se crea otra instancia de la clase **ObInterprete** con el Objeto respuesta.

g).-Se crea otra instancia de la clase **MenInterprete,** no para representar un nuevo mensaje sino para mostrar una línea de respuesta (cuyo origen es el objeto receptor original y su destino es el lugar donde aparecerá posteriormente el objeto respuesta). Esta línea representa la generación de la respuesta que realiza el objeto original.

h).-Se le envía un mensaje **dibújate** al objeto respuesta.

i).-Se les envía mensajes húndete a aquellos objetos ya dibujados que por no tener identificador deben desaparecer. (Recolector de basura).

j).-Se borran todas las instancias de **ObInterprete** y **MenInterprete** que ya no se requieran.

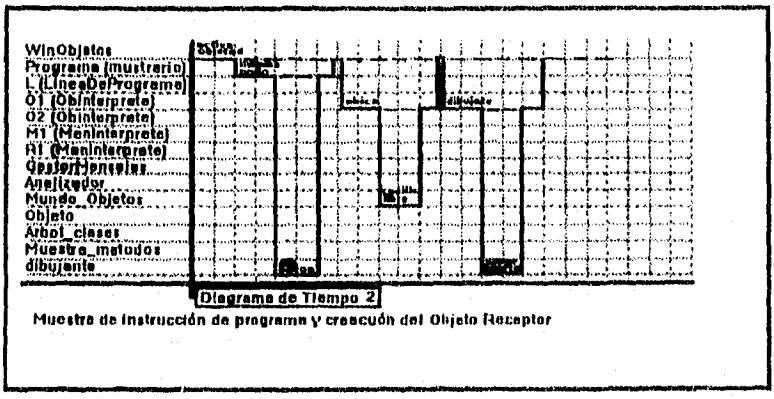
k).-Se repiten los pasos a-j, tantas veces como líneas de programa existan.

Profundizando un poco en los funcionamientos de los métodos esenciales podemos hablar de los Objetos y sus mensajes y de las Clases, considerando estos términos desde la perspectiva del usuario y no del programador.

Las instancias de la clase **Línea_de_Programa** al ser construidos reciben la cadena que representa la instrucción del programa en lenguaje OO (Smalltalk) y cuenta con los métodos para responder cuando se le consulte acerca del Objeto receptor, el mensaje a este enviado, los posibles argumentos (en esta versión, un argumento como máximo) y el identificador (la variable) opcional que etiqueta al objeto respuesta.

Las instancias de la clase **ObInterprete** se encargan de la representación gráfica del objeto, al crearse, reciben al objeto mismo (entero o carácter para esta versión) y lo almacena como atributo del mismo, así como el posible identificador. Este objeto cuenta con los métodos **dibújate**, **húndete** y **ubica** para decidir que posición del mundo de los objetos le corresponderá. Para ello, se vale de la información que intercambia con el objeto de la clase **MundoObjetos** a través del envío de los mensajes **casilla**, **casillaLibre** y **ocupa**. Además, el objeto de la clase **ObInterprete** consulta a la instancia de **MundoObjetos** para conocer cual es la ubicación desocupada más cercana a ella y con ello poder enviar la línea de respuesta a ese lugar.

El diagrama de tiempo 2 ilustra la representación de la instrucción y la creación del objeto receptor:

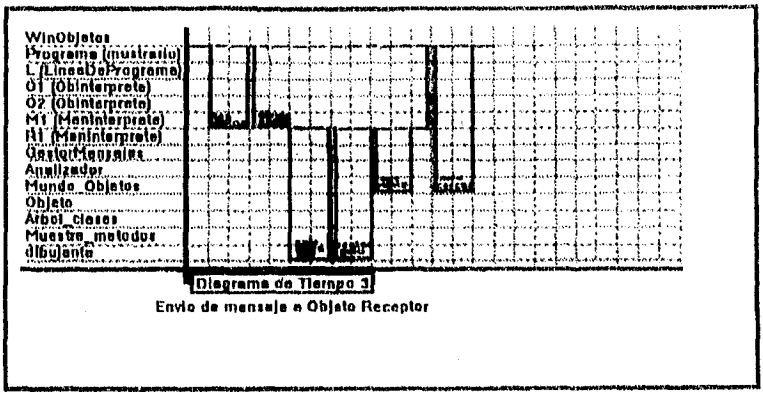


Muestra de instrucción de programa y creación del Objeto Receptor

El anterior es un diagrama a mayor detalle de los tres primeros mensajes enviados por **WinObjetos:**

Las instancias de la clase **MenInterprete** tienen a su cargo la representación gráfica tanto de los mensajes como de las líneas de respuesta que un objeto (desde el punto de vista del usuario) emite al efectuar uno de sus métodos. Para lograr lo anterior se interactua con las instancias de las clases **MundoObjetos** y **Dibujante.**

MenInterprete recibe los mensajes **TenDatos** y **dibujaEnvio** como se ilustra en el diagrama de tiempo tres:



La instancia de la clase **GestorMensajes** recibe de **WinObjetos** el nombre del Objeto, del mensaje y del posible argumento. Si todo lo descrito a continuación marcha normalmente, esta instancia devuelve un Objeto Respuesta.

GestorMensajes envía a la instancia de la clase **CAnalizador** (Analizador de clases) el nombre del objeto y recibe el nombre de la clase a la que pertenece. **GestorMensajes** realiza una búsqueda de métodos, inicialmente en la clase a la que pertenece el objeto buscando coincidencia con el mensaje, si no tiene éxito, busca en la clase padre y así sucesivamente hasta llegar a la clase más alta de la jerarquía o hasta el hallazgo.

Las Clases

La clase **Arbol_Clases** es una estructura arbórea que recibe los mensajes **agrega** para adicionar una nueva clase y **agrega_mensaje** para incluir un nuevo mensaje en una clase. Esto se efectúa una sola ocasión. Esta clase cuenta además con métodos para realizar todo el despliegue del árbol, de cada una de las clases o las ligas entre ellos, así como también el borrado de todo lo anterior. Para ello se vale de la clase **Dibujante.**

Arbol_Clases cuenta con la información tanto de las clases como de los mensajes que pertenecen a las clases.

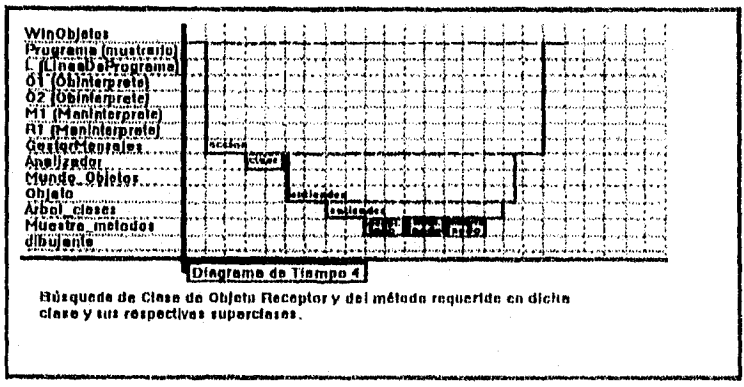
Para concluir el análisis a este nivel de abstracción, la descripción del funcionamiento de la interfaz con el usuario ilustrará y complemetara los apartados anteriores.

La interfaz con el usuario

El laboratorio gráfico básico de programación orientada a objetos se vale de la biblioteca de Objetos de Windows (OWL) y de dos clases desarrolladas adicionalmente: la clase **Dibujante** y la clase **Muestrario.**

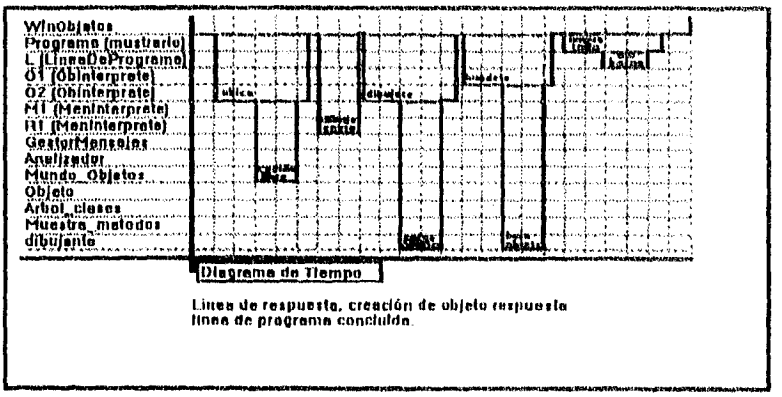
La instancia de la clase **Dibujante** es un objeto servidor que se encarga de toda la representación gráfica en pantalla.

Una instancia de la clase **Muestrario** despliega el programa OO a analizar, cada instrucción en una celda del muestrario, cuenta con los métodos para iluminar cada celda conforme se analiza la instrucción en ella desplegada. Por otro lado, cuando **GestorMensajes** está buscando coincidencia de mensaje y método en cada clase, se crea una instancia de **Muestrario** para cada clase, con ello se despliegan todos los mensajes en la clase contenidos y los ilumina secuencialmente, representando así la búsqueda del método.



Búsqueda de Clase de Objeto Receptor y del método requerido en dicha clase y sus respectivas superclases.

El diagrama de tiempo cuatro es una vista a detalle del mensaje **Acción** enviado por **WinObjetos**, mientras que el diagrama de tiempo cinco que a continuaci{on se muestra, ilustra los últimos cinco mensajes que **WinObjetos** envía.



Análisis a detalle de los objetos, los mensajes y la herencia entre las clases.

Como se dijo en el apartado anterior, el Gestor de Mensajes recibe de la ventana principal (**WinObjetos**), un mensaje con la instrucción del programa OO "desarmada" en Objeto, Mensaje y Argumento y debe regresar un objeto respuesta.

Para lograr lo anterior, **GestorMensajes** consulta al analizador de clases (**CAnalizador**) enviándole el objeto (tan sólo la representación del mismo) y recibiendo la clase a la que pertenece. Una vez que **GestorMensajes** sabe a qué clase pertenece el elemento de texto que representa a un objeto, entonces crea el objeto o lo instancia de dicha clase.

Todos los métodos instanciados de la manera anterior cuentan con el método **entiendes.** EL **GestorMensajes** entonces, consulta al objeto para saber si entiende el mensaje que se le intenta enviar, si responde afirmativamente entonces se le envía el mensaje y se espera la respuesta; de lo contrario, se presenta un mensaje de error en tiempo de ejecución (mensaje no entendido).

El mensaje **entiendes** que recibe cualquier objeto, delega esa consulta a **Arbol_Clases** de la siguiente manera:

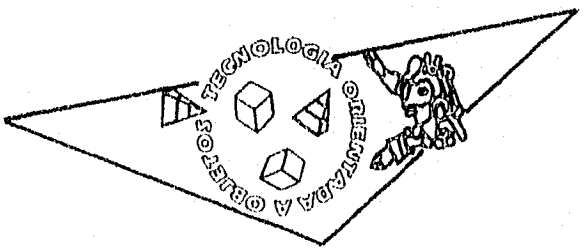
entiendes es un mensaje polimórfico que puede recibir de una forma cualquier objeto del subconjunto del modelo SmallTalk considerado y de dos formas un objeto de la clase **Arbol_Clases**.

Arbol_Clases cuenta con un método público **entiendes** que es enviado con tres argumentos, uno de ellos es una cadena. Este método detecta si existe una clase con ese nombre (la cadena) y regresa el

identificador de la misma en caso de éxito, y luego se envía un mensaje a sí mismo **entiendes**, éste último privado.

El método privado **entiende** de **Arbol_Clases** recorre todos los métodos con los que cuenta la clase en cuestión y regresa respuesta de éxito si encuentra el método buscado (incluido como otro argumento del mensaje). De no ser así, recorre todas las clases en la jerarquía ascendente en su búsqueda para lograr una respuesta.

V CONCLUSIONES



V CONCLUSIONES

El presente trabajo planteó como objetivo la creación de un sistema computacional didáctico que colabore en el proceso enseñanza-aprendizaje de la programación orientada a objetos. Para lograr lo anterior, se efectuó además del estudio minucioso de los conceptos relacionados con este modelo, una investigación detallada de las metodologías, contenidos y procesos involucrados en la enseñanza de este paradigma.

Se propone el uso del sistema descrito, en cursos introductorios de tecnología orientada a objetos sólo después de conocer los conceptos básicos o como inicio de un curso de programación orientada a objetos con un lenguaje OO puro. Además se ha utilizado en conferencias sobre este tema con bastante éxito.

De 1992 a 1995 se han impartido por parte del autor, cursos de tecnología orientada a objetos tanto escolarizados como abiertos pero sin llegar todavía a cincuenta alumnos.

Desde 1993 se han utilizado prototipos del sistema que aquí se presenta como apoyo a la exposición de algunos temas de manera controlada. 40 estudiantes participaron en la prueba, se les aplicó una evaluación acerca del entendimiento de los conceptos de modelo, a continuación se les expuso al contacto con la herramienta didáctica (estudiantes tan sólo como espectadores ya que no interactuaron) y finalmente se les aplicó la misma evaluación inicial.

Los resultados se presentan a continuación:

Mejoraron su calificación con un incremento mayor a 20 %	61%
Mejoraron su calificación entre un 5 y un 20 %	8%
Su calificación no varió en más de un 5%	30%
Su calificación experimentó un decremento mayor a 5%	1%

Considerando lo reducido de la muestra participante, no se puede afirmar de forma concluyente que esta herramienta didáctica sea o no útil; no obstante, es posible decir informalmente que el laboratorio OO presenta características que lo hacen parecer adecuado para mejorar el proceso de aprendizaje de esta tecnología.

En una primera versión, dado que se disponía de un compilador eficiente, se desarrolló este sistema didáctico en lenguaje Pascal, la versión definitiva para este trabajo recepcional está creada en C++ y se efectúa en ambiente Windows.

Cabe destacar que las versiones iniciales de este sistema fueron desarrolladas en una versión de Pascal no orientado a objetos, por tanto, la

experiencia adquirida al efectuar con una metodología, un programa cuyo objetivo es la enseñanza de esa misma metodología, aunado al hecho de haber desarrollado los prototipos sin aplicar, en un principio esa misma filosofía de programación, es realmente enriquecedora.

Dado que la intención del Laboratorio OO es precisamente la de auxiliar en el aprendizaje, este "meta-sistema" puede servir no solo para la enseñanza de un lenguaje OO puro (Smalltalk) sino que al tener disposición del código fuente, puede consultarse la manera en que éste ha sido construido en un lenguaje OO híbrido (C++).

Con base en los resultados de este trabajo se pueden proponer algunas líneas de investigación e hipótesis:

-Es factible desarrollar sistemas didácticos que apoyen la enseñanza no sólo de la programación sino de todo el proceso de construcción de sistemas orientados a objetos.

-Se puede mejorar el sistema actual (LabOO) presentando la interpretación de programas escritos en otros lenguajes, profundizando en la interpretación de cada instrucción componente de cada método analizado, ampliando el actual árbol de clases o permitiendo que el usuario lo modifique.

Puede concluirse finalmente que este trabajo ha tratado de obtener la información necesaria para prever los resultados que obtendrá el sistema desarrollado paralelamente y determinar el momento más oportuno para presentarlo al estudiante que lo utilizará durante un curso y los obstáculos de aprendizaje que en distintos grados pretende salvar.

Creemos que los objetivos se han logrado en buena parte y esperamos que sea de utilidad para el lector interesado en el fascinante mundo de la programación de computadoras.

MANUAL BASICO DEL USUARIO

Laboratorio Gráfico Básico de Programación Orientada a Objetos
Manual básico del Usuario

El laboratorio gráfico básico de Programación Orientada a Objetos es un sistema interactivo cuyo objetivo consiste en facilitar el entendimiento de los conceptos fundamentales de la programación orientada a objetos. En lo particular, esta herramienta didáctica trata de apegarse en la medida de lo posible a los conceptos reconocidos como puros de la tecnología orientada a objetos

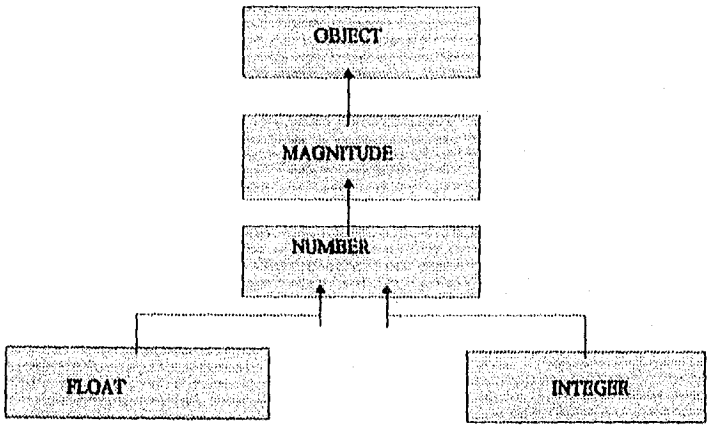
CONSIDERACIONES BASICAS

El sistema no pretende ser exhaustivo y dado que sus objetivos contemplan la ilustración de los conceptos básicos de programación orientada a objetos se considera tan sólo un subconjunto del árbol de clases total de Smalltalk y de los métodos aceptados por cada clase.

Dado que la descripción de los medios nos indica que el sistema será utilizado con una computadora de uso generalizado, entonces se optó por un interprete de Smalltalk utilizable en este tipo de máquina.

Se considera Smalltalk V de Digitalk.

El subárbol de clases que nuestro sistema contendrá es el siguiente:



REPRESENTACION GRAFICA OO

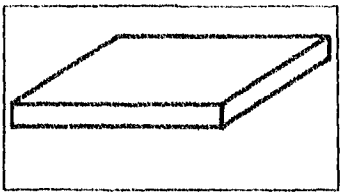
Se propone una representación gráfica de lo que conceptualmente suceda cuando se ejecuta un programa orientado a objetos. Es una representación muy particular que consiste en lo siguiente:

LINEA DE PROGRAMACION

Una línea de PROGRAMACION en un lenguaje orientado a objetos debe interpretarse como sigue.

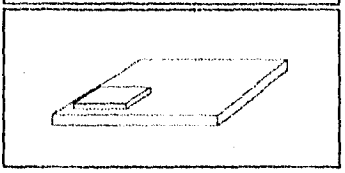
- Existe un objeto receptor, existe un mensaje dirigido al objeto receptor y existen potencialmente otros objetos que sirven como argumentos a dicho mensaje.

REPRESENTACION



-Existe un mundo único de objetos que puede contener hasta m x n objetos simultáneamente. En el mundo de los objetos, nacen objetos, envían y/o reciben mensajes, responden y o reciben respuestas de otros objetos. finalmente desaparecen los objetos.

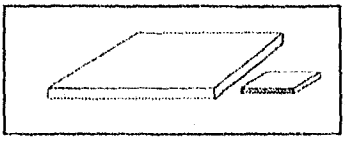
-Un objeto es un habitante del mundo de los objetos.



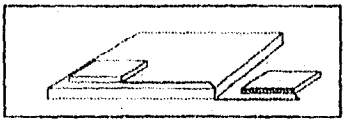
-La creación de un objeto se presenta por su surgimiento en el mundo de los objetos.

-La destrucción de un objeto se presenta de manera inversa a la de su creación.

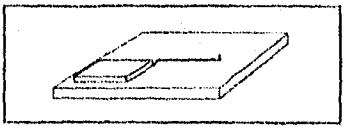
-Un objeto espontaneo es aquel que se crea en el mundo de los objetos por si sólo como consecuencia de ser referido en la linea de programación como objeto receptor o como objeto argumento del mensaje.



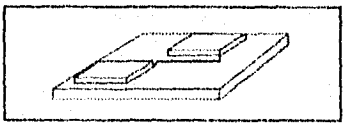
-Existe un ente externo al sistema que es quien escribe el programa, este programador es representado por un Objeto externo, objeto fuera del mundo de los objetos.



-El mensaje se representa por una línea que sale del objeto emisor y que concluye en el objeto receptor.



-Existe una línea de creación que surge de un objeto que ha recibido un mensaje y como respuesta crea un nuevo objeto



-Un objeto respuesta es un objeto con las mismas características de los demás objetos pero éste es creado por otro objeto como respuesta de éste último a un mensaje recibido.

REFERENCIA RAPIDA DE LABOO

En esta sección se describe de una manera rapida como debe iniciarse en el uso de LabOO. primeramente se muestra como se instala el sistema, luego se muestra como se inicia una sesión, a continuación se muestra el uso de cada una de las opciones del menu del sistema y finalmente se ilustra la ejecución básica del laboratorio interpretando una línea de programa escrita en lenguaje orientado a objetos.

Instalación de LabOO

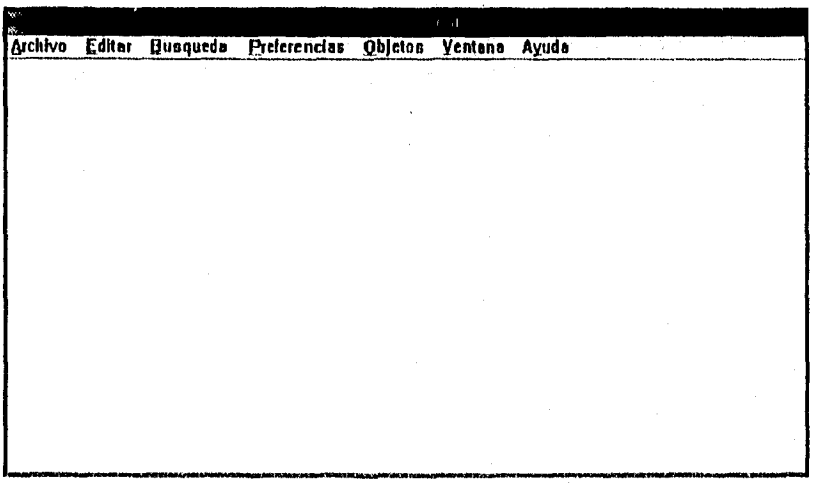
Cree en su disco duro un subdirectorio llamado **LaburaOO** (Desde DOS con MD LaboraOO)
Copie el contenido del disco **LabOO** el el subdirectorio creado (Con a>copy *.* c:\laboraOO)
Inicie una sesión con Windows
En Windows elija la opción **Archivo-Nueva**
En la ventana de diálogo que aparece elija **Grupo de programas** y **Aceptar**
Llene solo el dato superior que le pide la ventana de dialogo con **LabOO** y de **Aceptar**
Una vez que ya tiene la ventana de grupo **LabOO** elija **Archivo-Nuevo**
En la ventana de diálogo que aparece elija **Elemento de programa** y **Aceptar**
En la ventana de diálogo que aparece responda con:

Descripción: **LabOO**
Línea de comando: **LabOO**
Directorio de trabajo c:\ **LaboraOO**

El resto de la ventana no lo responda y de **Aceptar**
El Icono de **LabOO** aparecera en la ventana de grupo **LabOO**

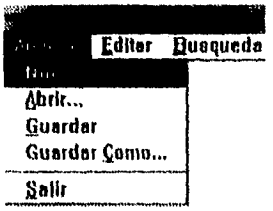
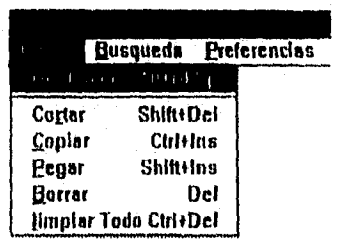
Entrada a LabOO

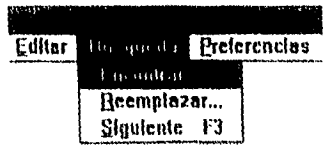
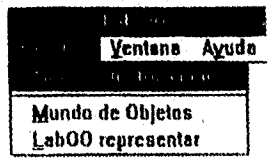
Para inicializar el sistema, señale con el apuntador del mouse el icono de "LabOO" presione un click y (enter) o presione doble click. LabOO presenta la siguiente pantalla principal:

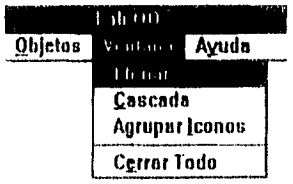


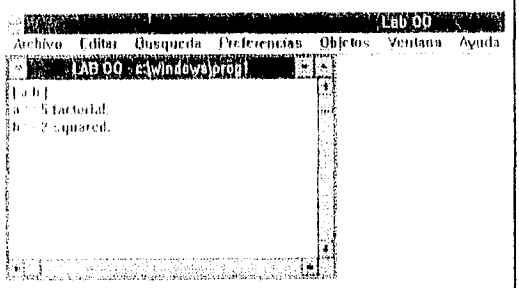
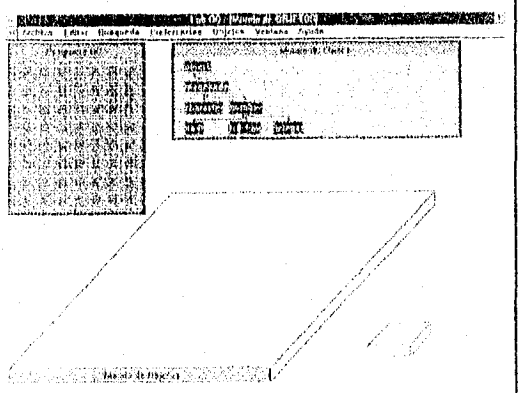
Menu principal

En esta sección se analiza cada una de las opciones del menú principal, en la columna de la izquierda se explica cada una de las subopciones que aparecen cuando se elije cada opción de dicho menú, en la columna de la derecha se ilustra la sección de pantalla correspondinte a la opción explicada.

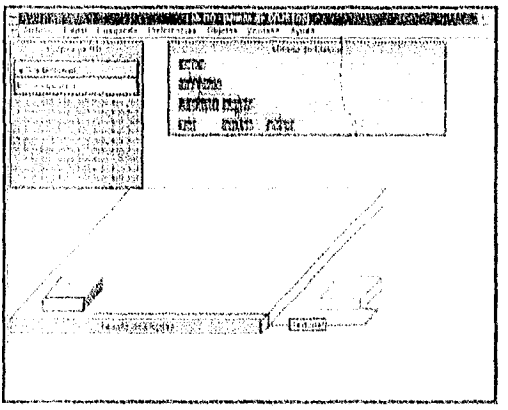
Archivo Las opciones que despliega esta ventana son: **Nuevo** Permite iniciar la escritura de un programa nuevo en lenguaje OO **Abrir** El usuario podrá activar un archivo que ha sido creado con anterioridad **Guardar** Almacena el archivo activo utilizando el mismo nombre y la ruta acceso especificada. **Guardar Como** Esta instrucción también almacena el archivo pero permite cambiar el nombre o la ruta de acceso. **Salir** instrucción que permite abandonar el sistema.	
Editar Este comando contiene las subopciones básicas de una editor de textos: **Deshacer** Regresa al estado que el editor guardaba antes de la última operación. **Cortar** Almacena en una posición de memoria la parte del texto seleccionada para poderla mover a otro lugar. **Copiar** Almacena en posición de memoria la parte del texto seleccionada para poderla duplicar. **Pegar** Despliega el texto elegido anteriormente con las opciones de cortar o coplar y lo coloca en la posición en donde se encuentre el cursor. **Borrar** Elimina el texto seleccionado. **Limpiar todo** Borra todo el texto de la ventana de edición.	

Búsqueda Este comando asiste al usuario cuando requiera: **Encontrar** Instrucción que asiste al usuario para ubicar un texto dentro de la ventana de edición. **Reemplazar** esta opción permite ubicar un texto y sustituirlo por otro. **Siguiente** Se utiliza cuando el usuario desee continuar la búsqueda de un texto o la búsqueda con remplazo, una vez que se ha encontrado se puede continuar con el rastreo.	
Preferencias Este comando contiene la opción: **Velocidad** Permite elegir al usuario la velocidad de representación, ya que LabOO puede ser ejecutado en computadoras de muy diversas velocidades y estas pueden compensarse con esta opción.	
Objetos Con este comando se realiza el control de todo lo que contiene la ventana de objetos. **Analizar instrucciones** Efectúa un análisis de cada una de las instrucciones del programa escrito en la ventana de edición y verifica tengan formato válido. **Mundo de objetos** Abre una ventana que muestra el mundo de los objetos, el mundo de las clases y un espacio para mostrar el programa (texto de la ventana de edición) y el objeto Usuario. **Representar** Es la opción escencial de LabOO. Se puede elegirla, si el programa de la ventana de edición ya fué analizado, con el fin de que represente gráficamente en pantalla lo que se efectúa al ser ejecutado dicho programa.	

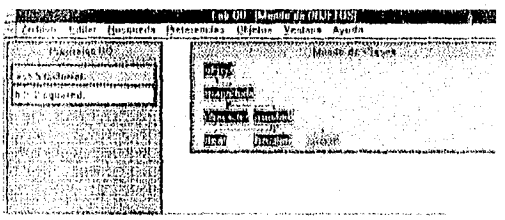
Ventana Permite manipular la ventana de LabOO. **Llenar** Reacomoda todas las ventanas abiertas de tal manera que ocupen la totalidad del área de trabajo de la ventana LabOO. **Cascada** Reacomoda las ventanas una detrás de otra dejando al descubierto la barra de título de cada una. **Agrupar Iconos** Acomoda los iconos en la ventana de forma ordenada. **Cerrar todo** Cierra la totalidad de ventanas abiertas desde LabOO a excepción de sí misma.	
Ayuda **Acerca de** Muestra la información general del programa de LabOO.	

USO DE LABOO	
ACCION	**RESULTADO**
Elija la opcion **Archivo-Nuevo**	Se Abre una nueva ventana de edicion y aparece el cursor al inicio de la misma
Escriba un programa en Smalltalk (subconjunto de Smalltalk considerado) Ejemplo "a b" a := 5 factorial. b := 2 squared.	
Elija la opcion **Objetos Analizar instrucciones**	se efectua un analisis de formato de cada linea del programa
Elija la opcion **Objetos Mundo de los objetos**	Se Abre una ventana en donde se mostrara la representacion grafica 

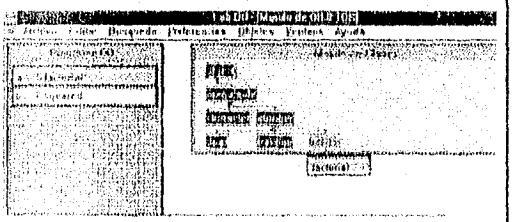
Del objeto externo (usuario) surge el mensaje con destino al objeto protagonista y muestra el nombre del mensaje.



El objeto al recibir el mensaje se autoanaliza para saber a que clase pertenece y en el mundo de las clases se manifiesta participando dicha clase.

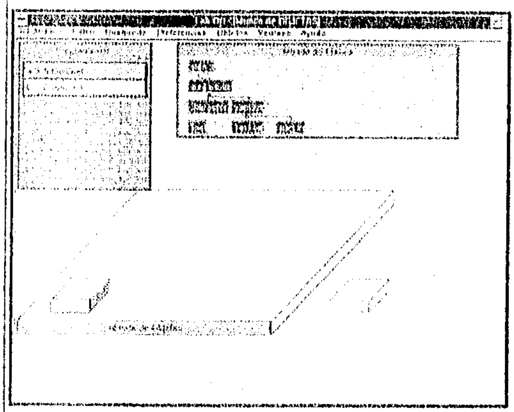


Se muestran todos los mensajes que es capaz de recibir un objeto de la clase a la que pertenece el objeto protagonista y se muestra uno por uno en inverso, simulando la búsqueda del método de nombre coincidente con el mensaje.

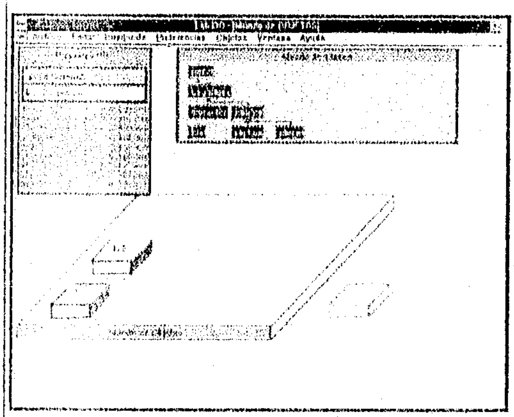


Si el método no fue encontrado entonces se coloca en reversa la superclase de la clase analizada y se repite la operación de búsqueda entre sus métodos; si este proceso continúa hasta la clase Object sin tener éxito se regresa el mensaje
"mensaje no entendido"

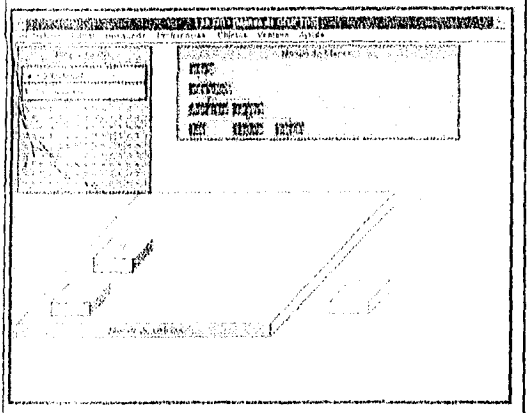
Si se encuentra el método se ejecuta y se obtiene el objeto respuesta y es representado en el mundo de los objetos.
Primeramente surge una línea del objeto receptor hacia un espacio cercano a dicho objeto.



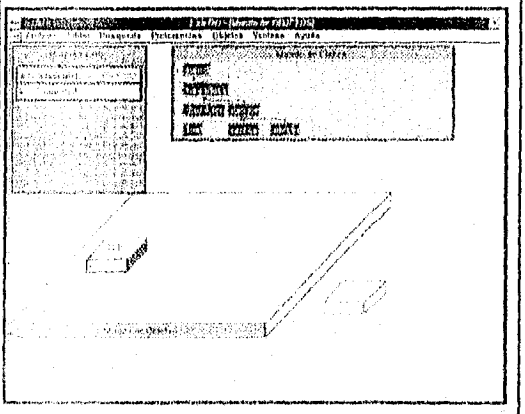
Surge el Objeto respuesta y en el aparece su valor



De igual forma la línea de respuesta x := 1 nuevo objeto queda asignado con la variable que le corresponde si esta contemplado en la especificación



Finalmente, desaparece el objeto receptor inicial ya que nadie lo esta referenciando



Este proceso se repite con cada una de las instrucciones del procedimiento

En **LabOO** también se toman en cuenta los siguientes aspectos importantes:

a) Cuando en el transcurso de un programa debe de crearse un objeto en el Mundo de los Objetos que previamente ya se creó y permanece vivo (es decir que está etiquetado por cuando menos una variable) este no se crea una vez más ya que no pueden existir dos objetos iguales.

b) Cuando en el transcurso de un programa se crea un objeto en el Mundo de los Objetos y este debe ser etiquetado con una variable que servía como identificador de otro objeto, éste último objeto desaparece si no existe otro identificador que lo etiquete.

El laboratorio gráfico básico de Programación Orientada a Objetos tiene el objetivo de fungir como complemento en la enseñanza de algunos de los conceptos de tecnología orientada a objetos y permitir un paso mas dinámico y veloz hacia la manera de pensar que esta metodología propone.

Es importante destacar que las motivaciones que produjeron la creación de este sistema, se fundamentan en la mejor intención de transmitir conocimientos y acelerar el avance de la educación y el desarrollo general de las ciencias de la computación en México

BIBLIOGRAFIA

REFERENCIAS:

[1] Ackroyd, Michael, Daun Dana. **Graphical notation for object-oriented design and programming.** JOOP 4(1) . Enero 1991. pp 18-28.

[2] Antebi, Morey. **Issues in Teaching C++.** JOOP 3(6) . Noviembre/Diciembre 1990. pp 11-21.

[3] Beck, Kent, Conningham, Ward. **A Laboratory for Teaching Object-Oriented Thinking.** OOPSLA '89 Proceedings. 1989. pp 1-6.

[4] Berard, Edward V. **Selecting and using consultants for Object-Oriented technology.** JOOP 6(7) Septiembre 1993. pp 48-53.

[5] Booch, Grady **Object Oriented Design with applications.** Editorial The Benjamin/Cummings Publishing Company. EUA. 1990-1993

[6] Buzzard G.D. Mudge T.N. **Object-Based Computing and the Ada Programming Languaje.** Computer. Marzo 1985. pp 11-19.

[7] Digitalk Inc. **Smalltalk, Tutorial and programming Handbook.** California, EUA. 1987

[8] Cardelli, Luca, Wegner, Peter. **On Understanding types, data abstraction and polimorphism, Computer Surreys.** 17 (4) Diciembre 1983. pp 472-522.

[9] Conway, Jane. Case Study: **GE facilitates the transition to Object-Oriented programming.** JOOP 6(3). Marzo/Abril 1993. suplemento pp 4-9.

[10] Cox, Brad J. Novobilski, Andrew J. **Programación Orientada a Objetos, un enfoque evolutivo.** Editorial Addison-Wesley/Diaz de Santos. Delaware EUA 1993.

[11] D'Souza, Desmond. **Teacher! Teacher**. JOOP 5(5). mayo 1992. pp 12-17.

[12] D'Souza, Desmond. **Flattenig the learning curve.** JOOP 5(7). Julio/Agosto 1992. pp 14-19.

[13] D'Souza, Desmond. **The learning curve: model projection.** JOOP 5(8). Septiembre 1992. pp 55-58.

[14] D'Souza, Desmond. **Navigating those learning curves.** JOOP 5(7). Octubre 1992. pp 21-25.

[15] D'Souza, Desmond. **Starting at the top.** JOOP 6(1). Enero 1993. pp 12-16.

[16] D'Souza, Desmond. **An educated look at education.** JOOP 6(3). Marzo/Abril 1993. pp 40-46.

[17] D'Souza, Desmond. **What on earth happened to the globalls?.** JOOP 6-(4), Mayo 1993. pp. 30-33.

[18] D'Souza, Desmond. **The cost of object education.** JOOP 7(6). Junio 1994. pp 62-64.

[19] Floyd, Michael. **Comparing Object-Oriented languages.** Dr Dobb's Journal. Octubre 1993. pp 104-125.

[20] Gibson, Elizabeth. **Flattening the learning curve: educating object-oriented developers.** JOOP 4(2). Febrero 1991. pp 24-29.

[21] Hinnant, David, **Justifying Object Technology** Object Magazine 5(3), Junio 1995, pp 70-74.

[22] Joyanes, A. Luis **Turbo C++ manual de bolsillo.** Editorial Mc Graw Hill. Madrid España 1992.

[23] Katrib Mora, Miguel **Programación Orientada a Objetos a través de C++ y Eiffel.** V Escuela de Invierno en temas selectos de la computación. México Octubre 1994.

[24] Katrib Mora, Miguel **Programación Orientada a Objetos en C++** INFOSYS. México 1994.

[25] Kempf , Renate, Steelzner Marilyn. **Teaching Object-Oriented Programming With the KEE System.** OOPSLA '87 Proceedings. 1987 pp. 11-25 1987.

[26] Korson, Tim. et al **Managing the Transition to Object-Oriented Technology.** OOPSLA '91 Proceedings. 1991 . pp 355-359.

[27] Lieberherr, Karl J., Riel, Arthur J. **Contributions to teaching Object-Oriented Design and Programming.** OOPSLA '89 Proceedings. 1989. pp.11-22

[28] Liu, Chamond, et al. **What Contributes to Successful Object-Oriented Learning?.** OOPSLA '92 Proceedings. 1992 . pp 77-86.

[29] Lorenz, Mark. **A return on your consulting investment.** JOOP 6(7) Septiembre 1993. pp 43-47.

[30] Love, Tom. **Flaying over the object barrier.** JOOP 6(3). Marzo/Abril 1993. suplemento pp. 10-12.

[31] Martin, James. O'Dell James. **Análisis y Diseño Orientado a objetos.** Editorial Prentice Hall. México 1992.

[32] Mckim, James C. Jr. **Teaching programming. And design.** JOOP 6(3). Marzo/Abril 1993. pp 32-39.

[33] Meyer, Bertrand. **Toward and Object-Oriented Curriculum.** JOOP 6(4). Mayo 1993. pp 76-81.

[34] Nerson, Jean-Marc. **Object-Oriented analisis and design, State of the Art and case studies.** IV escuela internacional de invierno en temas selectos de la computación, Mérida, Yucatán, 1993

[35] Ong, C. L, Tsai W.T. **Class and object extaction from imperative code.** JOOP 6(3). Marzo/Abril 1993. pp 59-68.

[36] Pratt, Terence W. **Lenguajes de Programación, Diseño e Implementación.** Editorial Prentice Hall 1987.

[37] Porter, Antony **Turbo C++ para windows.** Editorial Mc Graw Hill. México, D.F. 1994

[38] Rosson, Mary Beth. **Problem solution mapping in object-oriented design.** OOPSLA '89 proceedings.

[39] Schildt, Herbert **Aplique Turbo C++ For Windows.** Editorial Mc Graw Hill. España 1993.

[40] Shaw, Mary. **Larger scale systems require higher-level abstractions.** ACM, 1989.

[41] Tucker, Allen B. **Lenguajes de Programación.** Editorial Mc Graw Hill. México, D.F. 1988.

[42] Tsvi Bar, David. **Object-Oriented Education and Training in the 1990s .** JOOP 6(3). Marzo/Abril 1993. pp 24-31.

[43] Voss, Greg **Programación Orientada a Objetos, Una introducción.** Editorial Mc Graw Hill. México 1994.

[44] Wegner, Peter. **Learning the language.** BYTE, Marzo 1989.

[45] Wegner, Peter. **Concepts and paradigms of object-oriented programming.** OOPSLA '89 Proceedings 1989. pp. 6-20.

[44] Wegner, Peter. **Programming languages, the first 25 years.** IEEE transactions in computer, Diciembre 1993.

[47] Wiener , Richard Sincovec, Richard. **Programación en ADA.** Editorial Limusa 1989.

[48] Winblad, Ann L. et. al **Análisis y Diseño Orientado a Objetos.** Editorial Addison-Wesley/Diaz de Santos. Massachusetts, EUA. 1990.

[49] Wasserman, Anthony I. **Behavior and scenarios in object-oriented development.** JOOP 5(2). Febrero 1992. pp 61-64.

[50] Wiener, Richard, Pinson, Lewis. **OOP: An Academic Perspective.** JOOP 6(3). Marzo/Abril 1993. suplemento pp 13-15.

[51] Wiener, Richard, Sincovec, Richard **Programación en Ada,** Editorial Limusa, 1989, México, D.F.

[52] Wilson, David A. **Class Diagrams: A Tool For Design, Documentation and Teaching.** JOOP. 3(1). Enero/Febrero 1990, pp. 38-44.

[53] Wu C., Thomas **Teaching OOP to beginners.** JOOP 6(3). Marzo/Abril 1993. pp. 47-55.