UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

"ACTUALIZACIÓN PARA DOCENTES EN CÓMPUTO DEL NIVEL BACHILLERATO DE LA UNAM INCLUYENDO ASPECTOS DE ACTUALIDAD COMO APLICACIONES SOBRE INTERNET"

TRABAJO ESCRITO

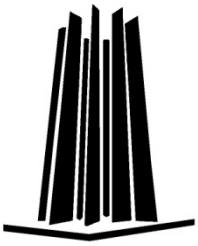
EN LA MODALIDAD DE SEMINARIOS Y CURSOS DE ACTUALIZACIÓN Y CAPACITACIÓN PROFESIONAL QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

S A N D R A D I E G O O R T I Z

ASESOR: M. EN C. MARCELO PÉREZ MEDEL



MÉXICO, 2007

En agradecimiento a nuestra institución y a las personas que directa e indirectamente hicieron posible la realización de este trabajo.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

ÍNDICE

INTRODUCCIÓN

En el 2003 se inicio en la entonces ENEP Aragón, ahora FES Aragón la primera generación del Diplomado enfocado a la actualización de los profesores de nivel bachillerato de la UNAM, llamado entonces "Actualización para docentes en cómputo del nivel bachillerato de la UNAM" creado con el objetivo de que los profesores tuvieran una herramienta de actualización en su vida Laboral y Académica. Este trabajo de tesis esta basado en dicho diplomado con la idea de servir como posibles apuntes para profesores que deseen introducirse en los temas tratados.

Al estudiar este trabajo analizaremos la abstracción de problemas, representarlos e implementarlos mediante algún lenguaje de programación. Se obtendrán conocimientos suficientes para la resolución de problemas por medio de diferentes paradigmas de programación, tales como programación estructurada o programación orientada a objetos y se entenderán los conceptos básicos para la creación de aplicaciones sobre Internet.

En el capítulo 1 se adquirirán las técnicas, conocimientos y habilidades para abstraer, analizar y solucionar problemas de tipo computable.

En el capítulo 2 se proporcionará a los lectores elementos para el manejo del lenguaje de programación Pascal y su uso para implementar soluciones incluyendo aspectos de actualidad como aplicaciones sobre Internet.

En el capítulo 3 se proporcionarán los conocimientos básicos teóricos y prácticos para el diseño y construcción de aplicaciones con interfaz gráfica de usuario, utilizando programación orientada a eventos.

En el capítulo 4 se proporcionarán los elementos para entender y utilizar la Programación Orientada a Objetos (POO), a través del lenguaje de programación JAVA.

En el capítulo 5 se revisarán las estrategias para la enseñanza y evaluación en áreas de cómputo.

Y finalmente en el capítulo 6 se brindarán los conocimientos para la creación de aplicaciones sobre Internet, utilizando el lenguaje de programación PHP.

OBJETIVO GENERAL

Proporcionar al docente que imparte asignaturas relacionadas con la informática y la computación, del nivel bachillerato de la UNAM, un espacio de actualización orientado a la programación, abarcando los aspectos didácticos, metodológicos, así como aspectos de actualidad como aplicaciones sobre Internet.

CAPÍTULO 1

1. SOLUCIÓN DE PROBLEMAS Y ALGORITMOS

1.1 Objetivo

Adquirir las técnicas, conocimientos y habilidades para abstraer, analizar y solucionar problemas de tipo computable.

1.2 Introducción

En este capítulo se pretende dar una explicación general acerca de la construcción de algoritmos a través de la abstracción y las técnicas de análisis, esto como una técnica de solución de problemas, Así como la relación que existe entre la construcción de algoritmos y la programación de computadoras.

La humanidad se ha pasado la vida buscando la solución de los diferentes problemas a los que se ha enfrentado a lo largo de los años y además de los mismos problemas que la propia humanidad se ha buscado, es por ello que los algoritmos resultan fundamentales como previa herramienta a la solución de un problema.

Un algoritmo entonces es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico. Pero para crear los algoritmos necesitamos primeramente obtener las principales características de un problema y esto se logra por medio de la abstracción la cual también abordaremos en este capítulo, y finalmente estudiaremos algunas técnicas de análisis las cuales nos servirán para crear los algoritmos que darán soluciones a los problemas planteados.

En el siglo XIX se produjo el primer algoritmo escrito para un computador, por medio del cual la humanidad comenzó a dar soluciones a infinidad de problemas.

1.3 Desarrollo

Para iniciar con el desarrollo del tema debemos conocer algunos conceptos, del que nos ocuparemos en particular tiene que ver con la siguiente pregunta: De los elementos presentes en un problema, ¿Cuántos son realmente importantes a trabajar para la solución del mismo?, sin duda esto dependerá del resultado deseado y de los recursos disponibles, pero indiscutiblemente no podemos trabajar con todos los elementos, porque la complejidad seria muy cercana a la "real", es por eso que trabajamos solo con aquellos componentes que presentan mayor significación para nuestro objetivo, pero trabajemos de manera formal este concepto.

1.3.1 Abstracción

ABSTRACCION, ABSTRACTO: el verbo griego que se traduce por 'abstraer', se usaba comúnmente para designar el acto de sacar algo de alguna cosa, separar algo de algo, privar a alguien de algo, poner algo aparte, arrancar algo de alguna cosa, etc.

ABSTRACCIÓN: es la operación intelectual que consiste en separar mentalmente lo que es inseparable en la realidad. La abstracción es el precedente o la lógica, el instrumento de la generalización, porque no podemos concebir los conocimientos generales, sin eliminar lo individual, es decir, sin abstraer. Toda idea generalizada es abstracta y posee realidad solo inteligible y no concreta, porque la abstracción no es función de la imaginación, sino propia de la razón discursiva que divide en la mente lo indivisible y separa lo inseparable, preparando el análisis a que excita la complejidad sintética de lo real. A lo abstracto se opone lo concreto. Es esto lo dado en la experiencia con todos sus elementos, el dato real o materia del conocimiento; mientras que lo abstracto es lo construido por el pensamiento, la forma, que no tiene mas limite que lo contradictorio. Con estas advertencias, se puede distinguir la realidad inteligible (propia de las abstracciones) de la realidad concreta (que es la que poseen los objetos). Al abstraer concebimos las cualidades independientes de las sustancias dentro de las cuales residen, aislando mentalmente los caracteres diferentes de las cosas para examinarlos aparte y cada uno en si mismo.

Si en la complexión de sucesos y en la multiplicidad de motivos que solicitan nuestra actividad es regla practica dividir para vencer, en la síntesis de la realidad se impone como exigencia distinguir y dividir (por medio de la abstracción) para conocer la complejas sinuosidades de lo concreto o, como se dice, el prisma de infinitas caras de la realidad. Es una división intelectual que

aplicamos a las ideas que tenemos de los objetos, al discernir sus elementos constitutivos.

La abstracción se emplea para preparar lo que los lógicos denominan método de eliminación, procedimiento en virtud del cual se van restando o abstrayendo de objetos aquellas cualidades que no les son adecuadas, y aun sirve de auxiliar poderoso para la definición, cuando se necesita recurrir a los grados imperfectos y entre ellos a la definición negativa que consiste en exponer lo que no es lo definido para dejar ante el pensamiento (por ministerio de la abstracción) aquellas notas o cualidades características de lo que se pretende definir. Abstrayendo, descubrimos las relaciones de semejanza que existen entre los objetos, y nos elevamos a la noción de lo que les es común (ideas generales), siendo digno de notarse que la abstracción prepara el uso de la generalización, dispone el análisis y es el requisito indispensable de la sistematización ordenada de nuestros conocimientos. Todo el conocimiento humano, en cuanto aspira a ser científico, tiene por base la abstracción, determinándose por tanto una relación directa entre el desarrollo de la abstracción y el progreso del pensamiento.

Es en efecto cada uno de nuestros sentidos es un instrumento natural de la abstracción, porque mediante ellos se perciben determinadas propiedades de la materia, con exclusión o abstracción de las demás (así es la vista sensible al color y no a la resistencia, en lo cual se funda después la distinción). Como conocemos empíricamente, imponiéndonos la misma experiencia la necesidad de abstraer, podemos afirmar que tenemos ideas abstractas, porque nuestra percepción nunca llega al fondo y al infinito detalle de las cosas, ni conoce el todo de nada. Conocemos pues siempre mediante abstracción y es esta una operación espontánea, natural y congénita con nuestro pensamiento. Es, además, reflexiva (verdadero auxiliar de la ciencia), cuando fijamos premeditadamente la atención en determinada propiedad, prescindiendo de las demás.

Por otro lado enfocándonos en los lenguajes de programación, estos han sido entonces las herramientas utilizadas mediante las cuales los diseñadores de lenguajes pueden implementar los modelos abstractos para formular algoritmos que terminarán resolviendo un problema. La abstracción ofrecida por los lenguajes de programación se puede dividir en dos categorías: abstracción de datos (pertenecientes a los datos) y abstracción de control (perteneciente a las estructuras de control).

Los lenguajes de programación, en palabras de Ben Ari "sirven para tender un puente entre el nivel de abstracción del mundo real y el de la computadora (hardware)". Mientras más alto sea el nivel de abstracción que un lenguaje de programación provee, esto es, mientras más se acerque al dominio del problema (mundo real), más fácil es entender los programas escritos, más fácil se hará la verificación y la reutilización de software. Este tipo de lenguajes, además, pueden facilitar el desarrollo de software escrito por expertos en el área del problema que se quiere resolver, sin necesidad de recurrir a programadores profesionales para programas relativamente sencillos.

Estos lenguajes complican la implementación de compiladores e intérpretes, ya que un nivel de abstracción alto dificulta el mapeo a un nivel de abstracción bajo; esto es, al lenguaje de máquina, además de los problemas de generar código eficiente.

Los diferentes paradigmas de programación han ido incrementando poco a poco su nivel de abstracción, comenzando desde los lenguajes máquina, lo más próximo al ordenador y más lejano a la comprensión humana; pasando por los lenguajes de comandos, los imperativos, la orientación a objetos (OO), la Programación Orientada a Aspectos (POA); u otros paradigmas como la Programación Declarativa, etc. El común denominador en la evolución de los lenguajes de programación, desde los clásicos o imperativos hasta los orientados a objetos, ha sido el nivel de abstracción del que cada uno de ellos hace uso.

Ejemplo: Abstracción aplicada a la Programación Orientada a Objetos.

Abstracción desde la programación Orientada a Objetos consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan. Así la abstracción encarada desde el punto de vista de la programación Orientada a Objetos expresa las características esenciales de un objeto, las mismas distinguen al objeto de los demás. Además de distinguir entre los objetos provee límites conceptuales. Entonces se puede decir que la encapsulación separa las características esenciales de las no esenciales dentro de un objeto. Si un objeto tiene mas características de las necesarias los mismos resultarán difíciles de usar, modificar, construir y comprender y esto dificultará la solución del problema.

1.3.2 Técnicas de análisis

Con el objeto de facilitar el diseño de algoritmos y la organización de los diversos elementos de los que se componen se utilizan algunas técnicas que muestran una metodología a seguir para resolver los problemas. Estas técnicas hacen que los programas sean más fáciles de escribir, verificar, leer y mantener. Algunas de las técnicas más conocidas son: Top Down y Botton Up.

TÉCNICA TOP DOWN: es una técnica para diseñar que consiste en tomar el problema en forma inicial como una cuestión global y descomponerlo sucesivamente en problemas más pequeños y por lo tanto, nos de una solución más sencilla.

La descomposición del problema original (y de las etapas subsecuentes), puede detenerse cuando los problemas resultantes alcanzan un nivel de detalle que el programador o analista pueden implementar fácilmente.

El problema se descompone en etapas o estructuras jerárquicas, de modo que se puede considerar cada estructura como dos puntos de vista: ¿lo qué hace?, y ¿cómo lo hace? Si se considera un nivel n de refinamiento, las estructuras se consideran de la siguiente forma:

nivel n : Vista desde el exterior.
"¿lo qué hace?"

Nivel n+1 : Vista desde el interior.
"¿cómo lo hace?"

Ejemplo de un diseño descendente (top-down) de un control de almacén:

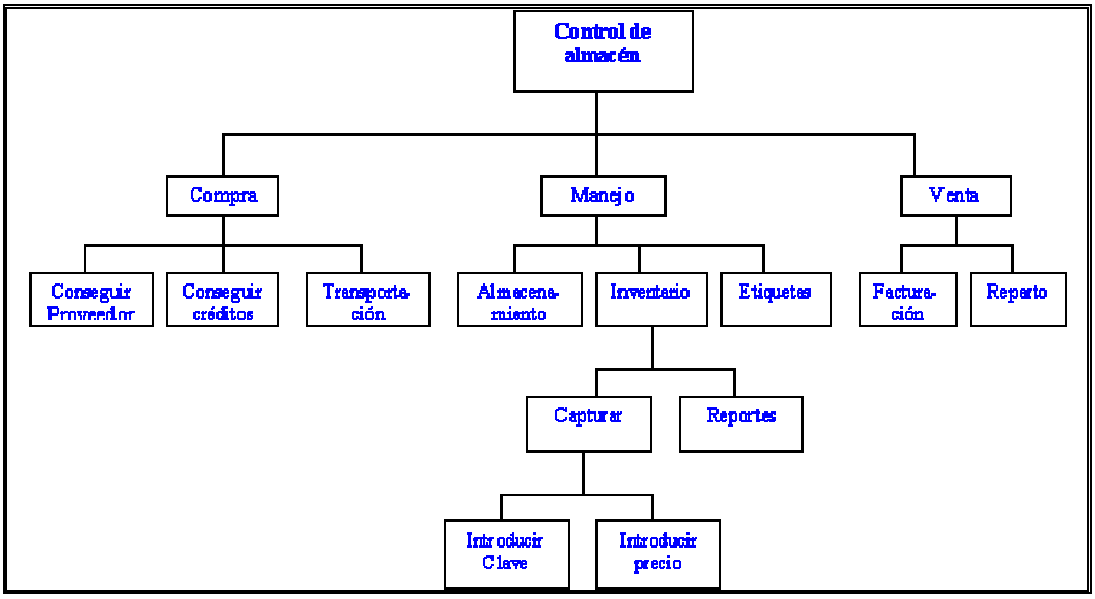


Figura 1.1 Ejemplo de un diseño descendente (top-down) de un control de almacén.

TÉCNICA BOTTON UP: esta técnica consiste en partir de los detalles más precisos del algoritmo completando sucesivamente módulos de mayor complejidad, se recomienda cuando ya se cuenta con experiencia y ya se sabe lo que se va a hacer.

Conforme se va alcanzando el desarrollo de módulos más grandes se plantea como objetivo final la resolución global del problema.

Este método es el inverso del anterior y es recomendable cuando se tiene un modelo a seguir o se cuenta con amplia experiencia en la resolución de problemas semejantes.

La técnica de Botton Up es frecuentemente utilizada para la realización de pruebas a sistemas ya concluidos.

La diferencia entre estas dos técnicas de programación se fundamenta en el resultado que presentan frente a un problema dado.

Ejemplo de un diseño ascendente (Botton Up):

Imagine una empresa, la cual se compone de varios departamentos (contabilidad, mercadeo, etc.), en cada uno de ellos se fueron presentando problemas a los cuales se le dieron una solución basados en un enfoque ascendente (Bottom Up): creando programas que satisfacían sólo el problema que se presentaba.

Por otro lado cuando la empresa decidió integrar un sistema global para suplir todas las necesidades de todos los departamentos se dio cuenta que cada una de las soluciones presentadas no era compatible la una con la otra, no representaba una globalidad, característica principal de los sistemas. Como no hubo un previo análisis, diseño de una solución a nivel global en todos sus departamentos, centralización de información, que son características propias de un diseño Descendente (Top Down) y características fundamentales de los sistemas; la empresa no pudo satisfacer su necesidad a nivel global.

Por lo tanto la creación de algoritmos es basado sobre la técnica descendente, la cual brinda el diseño ideal para la solución de un problema.

1.3.3 Solución de problemas

Enfocándonos en el medio computacional podemos decir que un problema es la especificación de una función entre un conjunto de datos y otro de resultados. El estudio de los problemas abarca dos grandes áreas: la de la teoría de la computabilidad y la de la teoría de la complejidad computacional.

De lo que podemos derivar que existen problemas irresolubles, es decir no toda función es computable y que existen también clases de complejidad, por lo tanto cada problema tiene una complejidad intrínseca.

A fin de resolver un problema utilizando sistemas de cómputo, debe seguirse una serie de pasos que permiten avanzar por etapas bien definidas hacia la solución.

Estas etapas son las siguientes:

- Definición del problema.
- Análisis de los datos.
- Diseño de la solución.
- Codificación.
- Prueba y depuración.
- Documentación.
- Mantenimiento.

DEFINICIÓN DEL PROBLEMA: está dada en sí por el enunciado del problema, el cual debe ser claro y complejo. Es importante que conozcamos exactamente "que se desea obtener al final del proceso"; mientras esto no se comprenda no puede pasarse a la siguiente etapa.

ANÁLISIS DE LOS DATOS: para poder definir con precisión el problema se requiere que las especificaciones de entrada y salida sean descritas con detalle ya que esto es un requisito para lograr una solución eficaz.

Una vez que el problema ha sido definido y comprendido, deben analizarse los siguientes aspectos:

- Los resultados esperados.
- Los datos de entrada disponibles.
- Herramientas a nuestro alcance para manipular los datos y alcanzar un resultado (fórmulas, tablas, accesorios diversos).

Una medida aconsejable para facilitar esta etapa consiste en colocarnos en lugar de la computadora deduciendo los elementos que necesitaremos para alcanzar el resultado.

DISEÑO DE LA SOLUCIÓN: Una computadora no tiene capacidad para solucionar problemas más que cuando se le proporcionan los sucesivos pasos a realizar, esto se refiere a la obtención de un algoritmo que resuelva adecuadamente el problema.

Esta etapa incluye la descripción del algoritmo resultante en un lenguaje natural, de diagrama de flujo o natural de programación.

Como puede verse, sólo se establece la metodología para alcanzar la solución en forma conceptual, es decir; sin alcanzar la implementación en el sistema de cómputo.

Tenemos que la información proporcionada constituye su entrada y la información producida por el algoritmo constituye su salida. Los problemas complejos se pueden resolver más eficazmente por la computadora cuando se dividen en subproblemas que sean más fáciles de solucionar.

CODIFICACIÓN: se refiere a la obtención de un programa definitivo que pueda ser comprensible para la máquina. Incluye una etapa que se reconoce como compilación. Si la codificación original se realizó en papel, previo a la compilación deberá existir un paso conocido como trascripción.

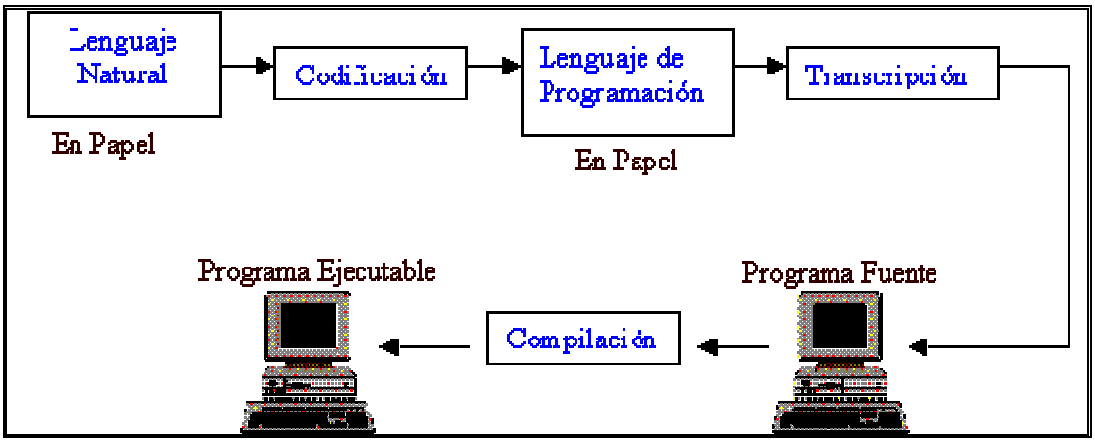


Figura 1.2 Ejemplo de los pasos a seguir para realizar la Codificación de un algoritmo.

Programa Fuente: esta escrito en un lenguaje de programación. (Pascal, C++,Visual Fox, Visual Basic, etc). Es escrito por el programador.

Programa Ejecutable: está en lenguaje máquina. Entendible por la máquina.

PRUEBA Y DEPURACIÓN: las pruebas que se le aplican son de diversa índole y generalmente dependen del tipo de problema que se está resolviendo. Comúnmente se inicia la prueba de un programa introduciendo datos válidos, inválidos e incongruentes y observando como reacciona en cada ocasión. El proceso de depuración consiste en localizar los errores y corregirlos en caso de que estos existan. Si no existen errores, puede entenderse la depuración como una etapa de refinamiento en la que se ajustan detalles para optimizar el desempeño del programa.

DOCUMENTACIÓN: debido a que el programa resultante en esta etapa se encuentra totalmente depurado (sin errores), se procede a la utilización para resolver problemas del tipo que dio origen a su diseño.
En vista de que esta utilización no podrá ser supervisada en todas las ocasiones por el programador, debe crearse un manual o guía de operación que indique los pasos a seguir para utilizar el programa.

MANTENIMIENTO: se refiere a las actualizaciones que deban aplicarse al programa cuando las circunstancias así lo requieran. Este programa deberá ser susceptible de ser modificado para adecuarlo a nuevas condiciones de operación. Cualquier actualización o cambio en el programa deberá reflejarse en su documentación.

1.3.4 Algoritmos

Un algoritmo es un conjunto ordenado y finito de instrucciones que conducen a la solución de un problema. En la vida cotidiana ejecutamos constantemente algoritmos. Por ejemplo, al instalar un equipo de sonido ejecutamos las instrucciones contenidas en el manual del equipo, este conjunto de instrucciones constituyen un algoritmo. Otro caso de algoritmo es el algoritmo matemático de Euclides para la obtención del máximo común divisor de dos números.

Si un algoritmo puede ser ejecutado por una computadora, se dice que es un algoritmo computacional; en caso contrario, se dice que es un algoritmo no computacional. Según esto, el algoritmo de Euclides es un algoritmo computacional; pero el algoritmo para instalar el equipo de sonido es un algoritmo no computacional. Para que un algoritmo pueda ser ejecutado por una computadora se necesita expresar el algoritmo en instrucciones comprensibles por la computadora, para esto se requiere de un determinado lenguaje de programación. Al algoritmo expresado en un determinado lenguaje de programación, se denomina programa. Puesto de otra manera, podemos decir que, un programa es la implementación o expresión de un algoritmo en un determinado lenguaje de programación siguiendo las reglas establecidas por el lenguaje elegido. En la Figura 2.3 que sigue se muestra la relación entre problema, algoritmo y programa.

Figura 1.3 Relación entre problema, algoritmo y programa.

Todo algoritmo debe tener las siguientes características:

- Debe ser preciso, es decir, cada instrucción debe indicar de forma inequívoca que se tiene que hacer.
- Debe ser finito, es decir, debe tener un número limitado de pasos.
- Debe ser definido, es decir, debe producir los mismos resultados para las mismas condiciones de entrada.
- Todo algoritmo puede ser descompuesto en tres partes:
- Entrada de datos.
- Proceso.
- Salida de resultados.

TÉCNICAS PARA LA FORMULACIÓN DE ALGORITMOS:

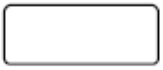
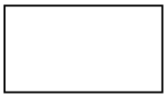
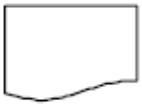
Las tres técnicas de formulación de algoritmos más populares son:

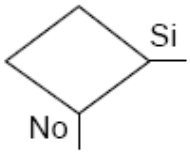
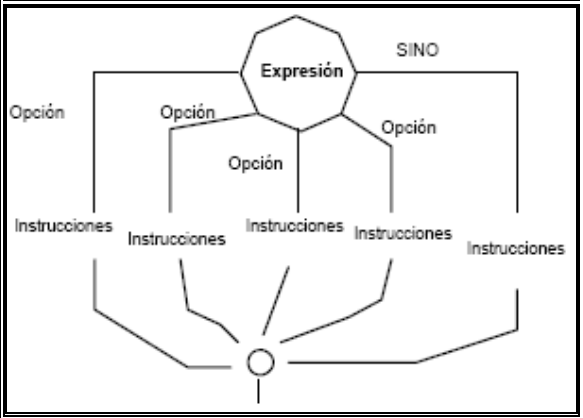
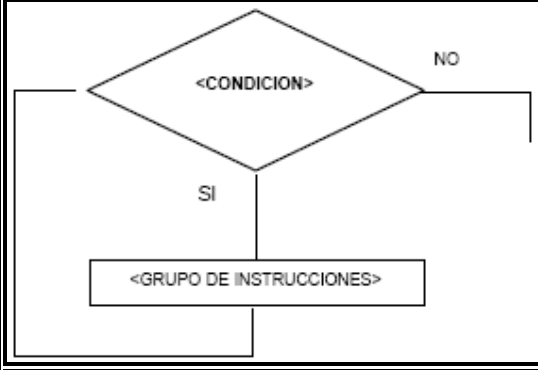
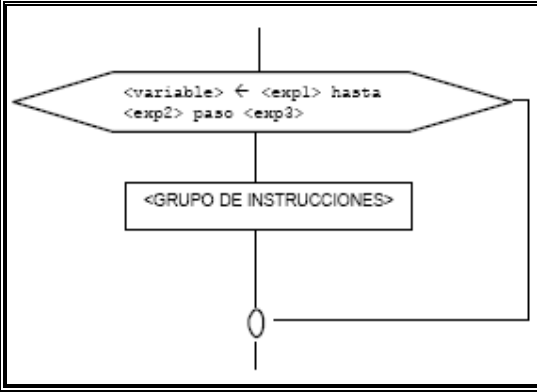
- Diagrama de flujo
- Pseudocódigo
- Diagramas estructurados

DIAGRAMAS DE FLUJO: se basan en la utilización de diversos símbolos para representar operaciones específicas. Se les llama diagramas de flujo porque los símbolos utilizados se conectan por medio de flechas para indicar la secuencia de operación. La simbología utilizada para la elaboración de diagramas de flujo es variable y debe ajustarse a un patrón definido previamente.

Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de como deben realizarse los pasos en la computadora para producir resultados.

Esta representación gráfica se da cuando varios símbolos (que indican diferentes procesos en la computadora), se relacionan entre si mediante líneas que indican el orden en que se deben ejecutar los procesos. Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI) y son los siguientes:

Símbolo	Descripción
	Indica el inicio y el final de nuestro diagrama de flujo.
	Indica la entrada y salida de datos.
	Símbolo de proceso y nos indica la asignación de un valor en la memoria y/o la ejecución de una operación aritmética.
	Indica la salida de información por impresora.
	Conector dentro de página. Representa la continuidad del diagrama dentro de la misma página.
	Conector fuera de página. Representa la continuidad del diagrama en otra página.

	Indica la salida de información en la pantalla o monitor.
	Símbolo de decisión. Indica la realización de una comparación de valores.
	Símbolo de Selección Múltiple. Dada una expresión permite escoger una opción de muchas.
	Símbolo del Mientras. Dada una expresión al principio de la iteración esta es evaluada; si la condición es verdadera realizará el ciclo, si es falsa la repetición cesará.
	Símbolo del Para. Esta estructura de control repetitiva se usa generalmente cuando se conoce de antemano el número de iteraciones.

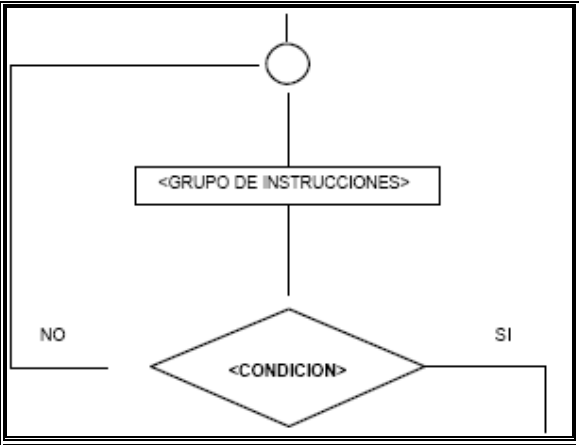
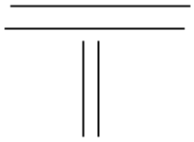
	Símbolo Repita Hasta. Funciona igual que la estructura Mientras, con la diferencia que al menos una vez hará el grupo de instrucciones y luego evaluará una condición. Si la condición evaluada es falsa continua dentro del ciclo y si es verdadera termina la iteración.
	Líneas de flujo o dirección. Indican la secuencia en que se realizan las operaciones.

Tabla 1.1 Símbolos de Diagramas de Flujo normalizados por el instituto norteamericano de normalización (ANSI).

Recomendaciones para el diseño de diagramas de flujo:

- Se deben usar solamente líneas de flujos horizontales y/o verticales.
- Se debe evitar el cruce de líneas utilizando los conectores.
- Se deben usar conectores sólo cuando sea necesario.
- No deben quedar líneas de flujo sin conectar.
- Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.
- Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.

Ejemplo de Diagrama de Flujo:

A continuación se muestra mediante el Diagrama de Flujo de la Fig. 2.4 los pasos a seguir para enviar un mensaje a través de la red.

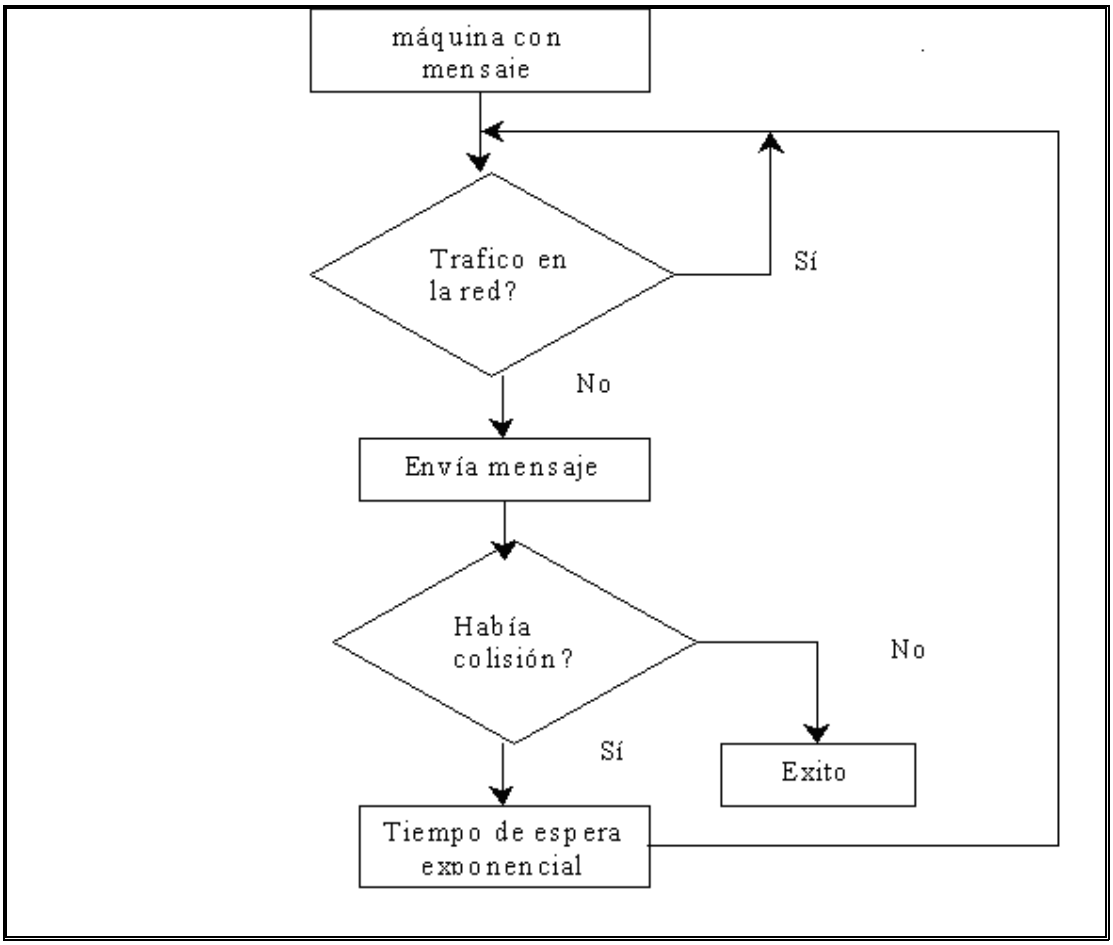


Figura 1.4 Ejemplo de Diagrama de Flujo

PSEUDOCÓDIGO: mezcla de lenguaje de programación y español (o ingles o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencial, el Pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El Pseudocódigo utiliza palabras que indican el proceso a realizar.

Ventajas de utilizar un Pseudocódigo a un Diagrama de Flujo:

- Ocupa menos espacio en una hoja de papel
- Permite representar en forma fácil operaciones repetitivas complejas
- Es muy fácil pasar de Pseudocódigo a un programa en algún lenguaje de programación.

La Tabla 1.2 muestra las palabras reservadas de Pseudocódigo:

Inicio y Fin: Por donde empieza y acaba el algoritmo. Begin /end : Pascal. { } : En C.
Sí <cond> Entonces <acc1> → If then else Sino <acc2>
Mientras <cond> /hacer → while do
Repetir / hasta → repeat until
Desde /hasta → for .. to
Según sea → Case Swith
Los comentarios van encerrados entre llaves.
Hay que utilizar la identación.

Tabla 1.2 Palabras reservadas de Pseudocódigo.

La estructura de un algoritmo en pseudocódigo es la siguiente:

```
Algoritmo <nombre del algoritmo>

Var
<nombre de la variable>: <tipo>

Inicio

<Instrucciones>

Fin
```

Tabla 1.3 Estructura de un algoritmo en pseudocodigo.

Ejemplo de un algoritmo en pseudocódigo:

Queremos hallar el producto de varios números positivos introducidos por teclado y el proceso termina cuando se meta un número negativo.

1. Iniciar la variable del producto.
2. Leer el primer número.
3. Preguntar si es negativo o positivo.
4. Si es negativo nos salimos y escribimos el producto.
5. Si es positivo, multiplicamos el número leído y luego leemos un nuevo número, y se vuelve al paso 3.

Algoritmo Producto:

```
Var
  P, num: entero
Inicio
  P ← 1
  Leer num
  Mientras num >=0 hacer
    P ← p*num
    Leer num
  Fin mientras
  Escribir p
Fin
```

DIAGRAMAS ESTRUCTURADOS (NASSI-SCHNEIDERMAN): El diagrama estructurado N-S también conocido como diagrama de chapin es como un diagrama de flujo en el que se omiten las flechas de unión y las cajas son contiguas. Las acciones sucesivas se pueden escribir en cajas sucesivas y como en los diagramas de flujo, se pueden escribir diferentes acciones en una caja.

La Figura 1.5 muestra la estructura básica de un Diagrama Estructurado:

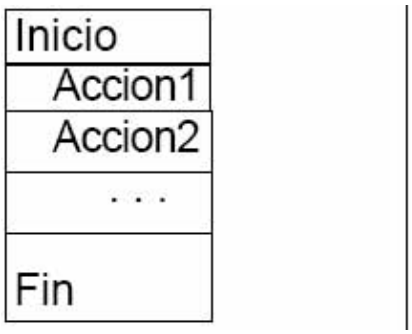
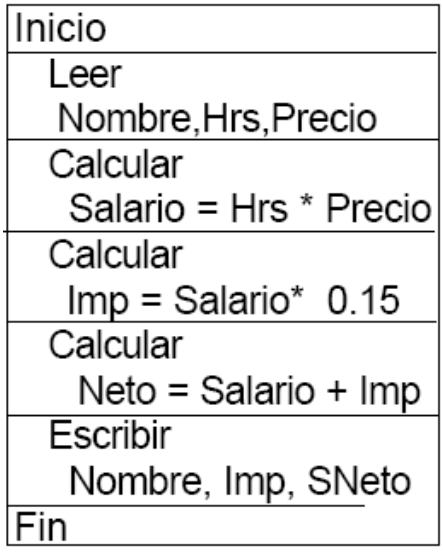


Figura 1.5 Estructura básica de un Diagrama Estructurado.

Un algoritmo se representa en la siguiente forma:



Los Diagramas Estructurados, son una técnica que permite formular algoritmos mediante una representación geométrica y de asignación de espacios de un bloque específico.

Este tipo de diagramas son semejante al diagrama de flujo, pero sin flechas y cambiando algo los símbolos de condición y repetición. Las cajas van unidas.

EJEMPLOS DE ALGORITMOS:

Observe el siguiente problema de tipo cotidiano y sus respectivos algoritmos representados en Pseudocódigo y en diagramas de flujos:

EJEMPLO 1: tengo un teléfono y necesito llamar a alguien pero no sé como hacerlo.

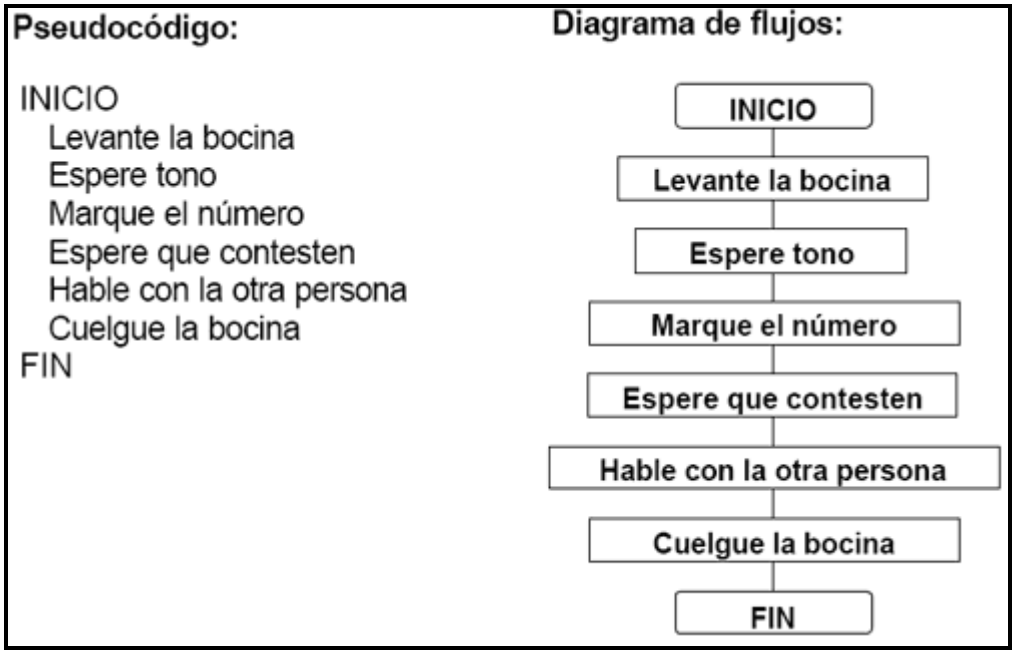


Figura 1.6 Problema de tipo cotidiano y sus respectivos algoritmos representados en Pseudocódigo y en diagramas de flujos.

Cuando se trabaja con algoritmos por lo general no se acostumbra a declarar las variables ni tampoco constantes debido a razones de simplicidad, es decir, no es camisa de fuerza declarar las variables.

EJEMPLO 2: escriba un algoritmo que pregunte por dos números y muestre como resultado la suma de estos. Use Pseudocódigo y diagrama de flujos.

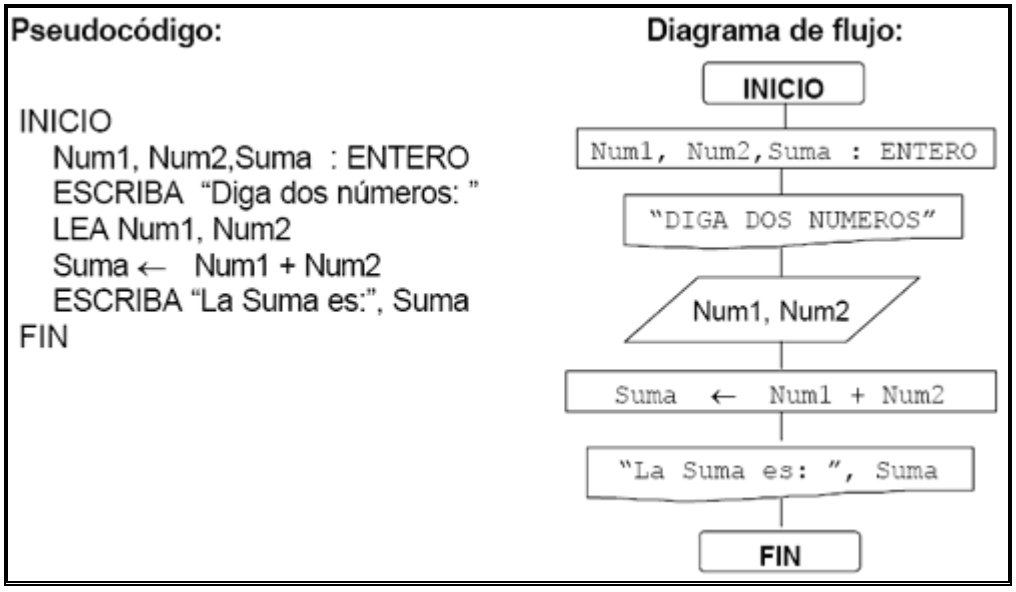


Figura 1.7 Pseudocódigo y diagrama de flujo del ejemplo 2.

EJEMPLO 3: escriba un algoritmo que permita conocer el área de un triángulo a partir de la base y la altura. Exprese el algoritmo usando Pseudocódigo y diagrama de flujos.

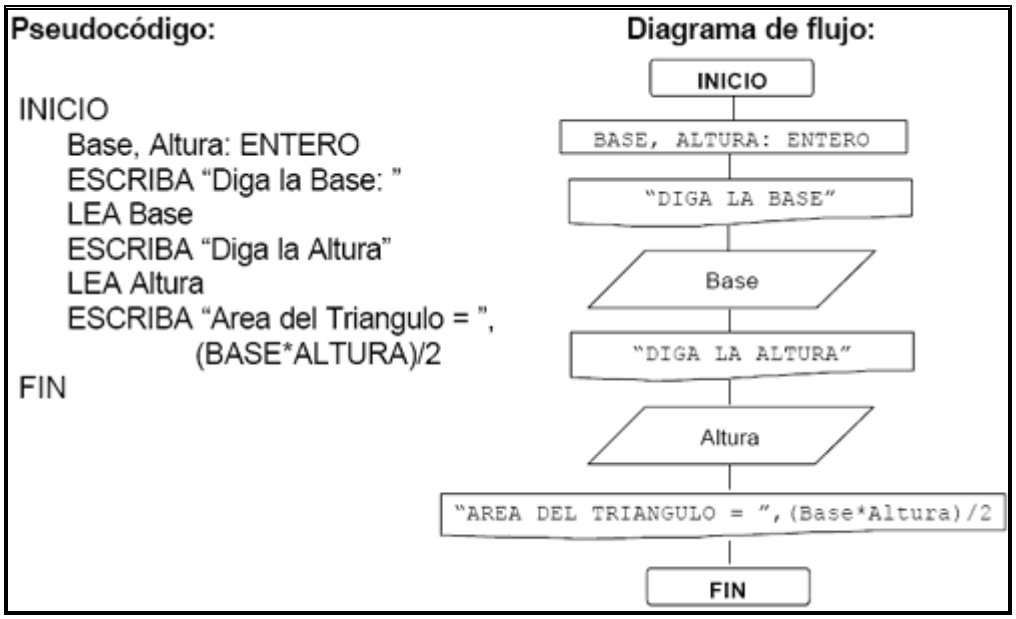


Figura 1.8 Pseudocódigo y diagrama de flujo del ejemplo 3.

EJEMPLO 4. Diagrama Estructurado:

Queremos hallar el producto de varios números positivos introducidos por teclado y el proceso termina cuando se meta un número negativo.

1. Iniciar la variable del producto.
2. Leer el primer número.
3. Preguntar si es negativo o positivo.
4 Si es negativo nos salimos y escribimos el producto.
5. Si es positivo, multiplicamos el número leído y luego leemos un nuevo número, y se vuelve al paso 3.

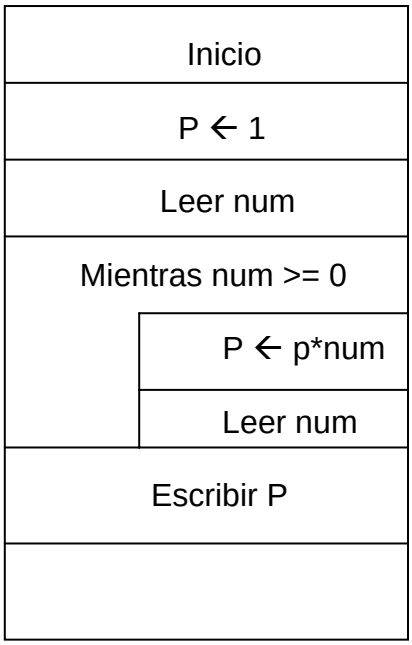


Figura 1.9 Diagrama Estructurado del ejemplo 4.

1.3.5 Computabilidad

La Teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que pueden ser resueltos con un algoritmo o equivalentemente con una máquina de Turing. La teoría de la computabilidad se interesa a cuatro preguntas:

¿Que problemas puede resolver una máquina de Turing?
¿Que otros formalismos equivalen a las máquinas de Turing?
¿Que problemas requieren máquinas más poderosas?
¿Que problemas requieren máquinas menos poderosas?

¿Que problemas puede resolver una máquina de Turing?

No todos los problemas pueden ser resueltos. Un problema indecidible es uno que no puede ser resuelto con un algoritmo aún si se dispone de espacio y tiempo ilimitado. Actualmente se conocen muchos problemas indecidibles, como por ejemplo:

- El Entscheidungs problem (problema de decisión en alemán) que se define como: Dada una frase del cálculo de predicados de primer orden, decidir si ella es un teorema. Church y Turing demostraron independientemente que este problema es indecidible.

- El Problema de la parada, que se define así: Dado un programa y su entrada, decidir si ese programa terminará para esa entrada o si correrá indefinidamente. Turing demostró que se trata de un problema indecidible.

- Un número computable es un número real que puede ser aproximado por un algoritmo con un nivel de exactitud arbitrario. Turing demostró que casi todos los números no son computables. Por ejemplo, el número de Chaitin no es computable aunque sí que está bien definido.

¿Qué otros formalismos equivalen a las máquinas de Turing?

Los lenguajes formales que son aceptados por una máquina de Turing son exactamente aquellos que pueden ser generados por una gramática formal. El cálculo Lambda es una forma de definir funciones. Las funciones que pueden se computadas con el cálculo Lambda son exactamente aquellas que pueden ser computadas con una máquina de Turing. Estos tres formalismos, las máquinas de Turing, los lenguajes formales y el cálculo Lambda son formalismos muy disímiles y fueron desarrollados por diferentes personas. Sin embargo, ellos son todos equivalentes y tienen el mismo poder de expresión.

Generalmente se toma esta notable coincidencia como evidencia de que la tesis de Church-Turing es cierta, que la afirmación de que la noción intuitiva de algoritmo o procedimiento efectivo de cómputo corresponde a la noción de cómputo en una máquina de Turing.

Los computadores electrónicos, basados en la arquitectura Von Neumann así como las máquinas cuánticas tendrían exactamente el mismo poder de expresión que el de una máquina de Turing si dispusieran de recursos ilimitados de tiempo y espacio. Como consecuencia, los lenguajes de programación tienen a lo sumo el mismo poder de expresión que el de los programas para una máquina de Turing y en la práctica no todos lo alcanzan. Los lenguajes con poder de expresión equivalente al de una máquina de Turing se denominan Turing completos.

Entre los formalismos equivalentes a una máquina de Turing están:

- Máquinas de Turing con varias cintas
- Máquinas de Turing con cintas bidimensionales (o una infinidad de cintas lineales)
- Máquinas de Turing con número limitado de estados y símbolos para la cinta
- Máquinas de Turing con solo dos estados
- Autómatas finitos con dos pilas
- Autómatas finitos con dos contadores
- Gramáticas formales
- Sistemas de correspondencia de Post
- Cálculo Lambda
- Funciones recursivas parciales
- Casi todos los lenguajes de programación modernos si dispusieran de memoria ilimitada
- Autómatas celulares
- El Juego de la vida de John Conway
- Máquinas de Turing no determinísticas
- Máquinas de Turing probabilísticas
- Computador cuántico

¿Qué problemas requieren máquinas más poderosas?

Se considera que algunas maquinas tienen mayor poder que las maquinas de Turing. Por ejemplo, una maquina oráculo que utiliza una caja negra que puede calcular una función particular que no es calculable con una maquina de Turing. La fuerza de cómputo de una maquina oráculo viene descrita por su grado de Turing. La teoría de cómputos reales estudia maquinas con precisión absoluta en los números reales. Dentro de esta teoría, es posible demostrar afirmaciones interesentes, tales como «el complemento de un conjunto de Mandelbrot es sólo parcialmente decidible».

En 1928 se había publicado un pequeño texto de lógica de Hilbert y Ackermann. El libro enfatizaba la lógica de primer orden. Los autores mostraron cómo las varias partes de las matemáticas podían ser formalizadas dentro de la lógica de primer orden y daban un conjunto sencillo de reglas de prueba para hacer las inferencias lógicas. Hacían notar que toda inferencia que podía ser realizada de acuerdo con las reglas de prueba era también válida, en el sentido de que en una estructura matemática en que todas las premisas son verdaderas, la conclusión es también verdadera.... Un problema levantado por Hilbert y Ackermann en su tratado era el problema decisorio, el problema de encontrar un algoritmo que determine si una inferencia propuesta determinada es válida. Este problema es equivalente a buscar un algoritmo para determinar si una conclusión particular puede derivarse de ciertas premisas usando las reglas de prueba Hilbert- Ackermann. El problema decisorio fue llamado "el principal problema de la lógica matemática", porque un

algoritmo para el problema decisorio podría, en principio, ser usado para contestar cualquier pregunta matemática: sería suficiente emplear una formulación en lógica de primer orden de la rama de las matemáticas pertinente a la cuestión bajo consideración. La atención de Alan Turing fue atraída hacia el problema decisorio... y pronto vio cómo resolver el problema en la negativa. Esto es, Turing mostró que no hay algoritmo alguno para resolver el problema decisorio. Las herramientas que Turing desarrolló para este propósito han resultado ser absolutamente fundamentales para la informática.

Si una solución positiva al problema decisorio llevaría a algoritmos para resolver todas las cuestiones matemáticas, entonces se sigue que si existe incluso un solo problema que no tiene solución algorítmica, entonces el mismo problema decisorio no puede tener solución algorítmica. Ahora bien, la noción intuitiva de algoritmo sirve perfectamente bien cuando lo que necesitamos verificar es que algún procedimiento propuesto de verdad constituye una solución positiva a un problema dado. Sin embargo, si permanecemos en este nivel intuitivo, no podríamos esperar probar que algún problema no tiene solución algorítmica. Para estar seguros de que ningún algoritmo funcionará, parece necesario hacer un reconocimiento de la clase de todos los algoritmos posibles. Esa es la tarea que se impuso Turing. Comenzó con un ser humano, quien realizaría los pasos sucesivos requeridos por algún algoritmo; esto es, Turing propuso considerar el comportamiento de un "computador". Aquí la palabra computador se refiere a una persona que realiza una computación; así era como Turing (y todo el mundo) usaba la palabra en 1935. Turing procedió por una secuencia de simplificaciones, cada una de las cuales podía verse que no planteaba ninguna diferencia esencial, para obtener su caracterización de computabilidad.

La teoría de la computabilidad y la no computabilidad es usualmente llamada la teoría de las funciones recursivas. Esta materia concierne la existencia de procedimientos puramente mecánicos para resolver varios problemas. Aunque la teoría es una rama de las matemáticas puras, es, por su pertinencia en relación con ciertas cuestiones filosóficas y la teoría de los computadores digitales, de interés potencial para los no matemáticos. La existencia de problemas absolutamente insolubles y el teorema de la incompletabilidad de Gödel están entre los resultados en teoría de la computabilidad que tienen significación filosófica. La existencia de máquinas universales de Turing, otro resultado de la teoría, confirma la creencia de aquellos que trabajan con computadoras digitales de que es posible construir un computador digital de propósito general en el cual se pueda programar (dentro de limitaciones de tiempo y memoria) cualquier problema que pudiera programarse para cualquier computador digital determinístico concebible. Este aserto se oye a veces en su forma reforzada: cualquier cosa que puede hacerse completamente precisa puede ser programada para una computadora digital de propósito general. Sin embargo, en esta forma el aserto es falso.

La gran importancia del concepto de recursividad general (o computabilidad de Turing) es que con este concepto se ha tenido éxito por primera vez en dar una

definición absoluta de una noción epistemológica importante, a saber sin dependencia de un formalismo determinado.

La creciente percepción de que una solución al problema decisorio produciría un procedimiento para solucionar muchos (si no todos) los problemas matemáticos significó que algunos matemáticos, especialmente von Neumann, se convencieran de que no podía haber tal solución; una creencia que llegó a ser casi una certidumbre cuando Gödel publicó [su famoso artículo en] 1931. Para fundar esa certidumbre, sin embargo, era esencial poner límites precisos a "eficazmente calculable". Esto (combinado con un interés natural por el cálculo mecánico) era lo que estaba en la atmósfera que Turing respiraba....

Turing mostró que el cálculo puede descomponerse en la iteración (controlada por un “programa”de operaciones concretas extremadamente simple; tan concreta que pueden fácilmente ser descritas en términos de mecanismos (físicos).

La computabilidad Turing fue una de tres maneras equivalentes de caracterizar exactamente las funciones para las cuales existen algoritmos que aparecieron en los años mil novecientos treinta. Los conceptos usados en las otras dos fueron la definibilidad lambda de Church-Kleene (desarrollada en la Universidad de Princeton en 1932 y 1933) y la recursividad general de Herbrand-Gödel (presentada por Gödel en sus conferencias de 1934 en el Instituto de Estudios Avanzados en Princeton). Su equivalencia la estableció Kleene en 1936.

1.4 Aplicaciones Generales

En realidad, en la vida cotidiana empleamos algoritmos en multitud de ocasiones para resolver diversos problemas. Ejemplos son el uso de una lavadora (se siguen las instrucciones), para cocinar (se siguen los pasos de la receta). También, existen ejemplos de índole matemática, como el algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para calcular el máximo común divisor de dos enteros positivos, o incluso el método de Gauss para resolver sistemas de ecuaciones.

1.5 Conclusiones

En conclusión podemos decir que para llegar a la solución o a una de las mejores soluciones posibles de un problema existe toda un metodología a seguir la cual es recomendable ya que nos puede llevar paso a paso a una solución fiable, además de darnos cuenta de que estamos rodeados de algoritmos que alguien en algún momento creo, y que por medio de ellos hemos llegado a una actualidad con cientos de problemas resueltos

CAPÍTULO 2

1. LENGUAJE DE PROGRAMACIÓN PASCAL

2.3 Objetivo

Proporcionar a los lectores elementos para el manejo del lenguaje de programación Pascal y su uso para implementar soluciones incluyendo aspectos de actualidad como aplicaciones sobre Internet.

2.4 Introducción

Pascal es un lenguaje de programación de alto nivel y propósito general, desarrollado por Niklaus Wirth, profesor del Instituto tecnológico de Zurich, Suiza. La primera versión preliminar apareció en 1968 y el primer compilador aparece a finales de 1970. Pascal se diseño para la enseñanza de la programación como una disciplina y superar las limitaciones que presentaban los lenguajes modulares (orientados a rutinas). Con el tiempo se ha convertido además en un estándar de los lenguajes de programación más usados.

Nació a partir del lenguaje de programación Algol al cual añade tipos de datos y simplifica su sintaxis. El nombre de Pascal fue escogido en honor al matemático Blaise Pascal.

Los sistemas programados en Pascal, utilizando la metodología de diseño y programación estructurada, son sistemas modulares, comprensibles y fáciles de modificar y de darles mantenimiento.

Sus características principales son:

- No es sensible al tamaño (mayúsculas y minúsculas no hacen diferencia en identificadores).
- Fuertemente tipeado. Se cuentan con muchos tipos de datos básicos, y se pueden crear tipos nuevos de manera sencilla.
- Esta diseñado considerando los conceptos de programación estructurada.

.

Existen varios compiladores de Pascal que están disponibles para el uso del público en general:

- Compilador GNU Pascal (GPC), escrito en C, basado en GNU Compiler Collection (GCC). Se distribuye bajo licencia GPL.
- Free Pascal está escrito en Pascal (el compilador está creado usando FreePascal), es un compilador estable y potente. También distribuido libremente bajo la licencia GPL. Este sistema puede mezclar código Turbo Pascal con código Delphi, y soporta muchas plataformas y sistemas operativos
- Turbo Pascal fue el compilador Pascal dominante para PCs durante los años 1980 y hasta principios de los años 1990, muy popular debido a sus magníficas extensiones y tiempos de compilación sumamente cortos. Actualmente, versiones viejas de Turbo Pascal (hasta la 5.5) están disponibles para descargarlo gratuito desde el sitio de Borland (es necesario registrarse)
- Delphi es un producto tipo RAD (Rapid Application Development) de Borland. Utiliza el lenguaje de programación Delphi, descendiente de Pascal, para crear aplicaciones para la plataforma Windows. Las últimas versiones soportan compilación en la plataforma .NET.
- Kylix es una nueva versión de Borland reiterando la rama de Pascal de sus productos. Es descendiente de Delphi, con soporte para el sistema operativo Linux y una librería de objetos mejorada. El compilador y el IDE están disponibles para uso no comercial. Actualmente este proyecto esta descontinuado.
- Proyecto Lazarus[1], el cual es un proyecto de software libre.

2.3 Desarrollo

Durante el desarrollo del tema se verán las caracteristicas principales del lenguaje de programación Pascal.

2.3.1 Estructura básica de un programa en Pascal.

Todos los programas comienzan con la palabra clave "Program", y un bloque de código es indicado con el "Begin"/"End". No hace diferenciación entre instrucciones o variables escritas en mayúsculas o minúsculas como hace el lenguaje de programación C. El punto y coma separa las declaraciones, y el punto sirve para indicar el final del programa o unidad. Para algunos compiladores la línea de Program es opcional.

En la Tabla 2.1 se muestra la descripción básica de un programa en Pascal, los detalles de cada parte se analizarán más adelante conforme se requiera.

Program nombre (input, output);	Nombre del programa, es opcional.
Uses Nombre de unidad;	Indicación de que bibliotecas precompiladas (Unidades) se cargarán.
Const Declaración de constantes;	Opcional.
Type Declaración de variables;	Opcional.

Procedure nombre (parámetros);
Begin
Cuerpo del procedimiento;
End;
	Declaración de los procedimientos y funciones.

Begin Cuerpo del programa; **End.**	Programa principal, nótese que termina en end. (end y un punto).

Tabla 2.1 Estructura de un programa en Pascal

El típico programa Hola mundo:

```
PROGRAM Holamundo (output);
BEGIN
WriteLn('¡Hola mundo!');
END.
```

Figura 2.1 Ejemplo de un programa en Pascal

2.3.2 Tipos de datos, operadores y funciones estándar.

TIPOS DE DATOS BÁSICOS

Los tipos básicos de Pascal son los que se muestran en la Tabla 3.2. En Pascal existe la posibilidad de crear fácilmente nuevos tipos de datos.

Nombre	Tipo	Descripción
Integer	Entero	Entre -32,768 y 32,767
Shortint	Entero corto	-128 a 128
Byte	Entero de 8 bits	0 a 255
Word	Entero 2 bytes	0 a 65535
Longint	Entero largo	-2*10^31 y 2*10^31
Char	Carácter	Una carácter ASCII
String	Cadena	Cadenas de texto ASCII
Boolean	Booleano	Verdadero o falso
Real	Real o de punto flotante	-2.9*10^39 y 1.7*10^

Tabla 2.2 Tipos de datos básicos en Pascal

OPERADORES

Los operadores sirven para combinar los términos de las expresiones.
En Pascal, se manejan tres grupos de operadores:

- ARITMÉTICOS
- RELACIONALES
- LÓGICOS

OPERADORES ARITMÉTICOS: Son aquellos que sirven para operar términos numéricos. Estos operadores podemos clasificarlos a su vez como:

- unarios
- binarios

Unarios: son aquellos que trabajan con UN OPERANDO. Pascal permite el manejo de un operador unario llamado: MENOS UNARIO

Este operador denota la negación del operando, y se representa por medio del signo menos (-) colocado antes del operando.

Por ejemplo:

Si x tiene asignado el valor 100, -x dará como resultado -100 ; esto es que el resultado es el inverso aditivo del operando.

Binarios: son los que combinan DOS OPERANDOS, dando como resultado un valor numérico cuyo tipo será igual al mayor de los tipos que tengan los operandos.

La Tabla 2.3 muestra los símbolos de los operadores binarios de Pascal así como los nombres de las operaciones que realizan.

Operador	Operación	Operandos	Ejemplo	Resultado
+	Suma	real , integer	a + b	suma de a y b
-	Resta	real , integer	a - b	Diferencia de a y b
*	Multiplicación	real , integer	a * b	Producto de a por b
/	División	real , integer	a / b	Cociente de a por b
Div	División entera	integer	a div b	Cociente entero de a por b
Mod	Módulo	integer	a mod b	Resto de a por b
Shl	Desplazamiento a la izquierda		a shl b	Desplazar a la izquierda b bits
Shr	Desplazamiento a la derecha		a shr b	Desplazar a la derecha b bits

Tabla 2.3 Operadores binarios de Pascal

Conviene observar lo siguiente:

1. Cuando los dos operandos sean del tipo integer, el resultado será de tipo integer.
2. Cuando cualquiera de los dos operandos, o ambos, sean del tipo real, el resultado será de tipo real.
3. Cuando, en la operación div, OPERANDO-1 y OPERANDO-2 tienen el mismo signo, se obtiene un resultado con signo positivo; si los operandos

difieren en signo, el resultado es negativo y el truncamiento tiene lugar hacia el cero.

Ejemplos:

7 div 3 = 2

(-7) div (-3) = 2

(-7) div 3 = -2

7 div (-3) = -2

15.0 div 3.0 = no válido

15 div (4/2) = no válido

La operación div almacena sólo la parte entera del resultado, perdiéndose la parte fraccionaria (truncamiento).

4. La operación MODULO está definida solamente para OPERANDO-2 positivo. El resultado se dará como el entero no negativo más pequeño que puede ser restado de OPERANDO-1 para obtener un múltiplo de OPERANDO-2 ;

   Ejemplo:

   6 mod 3 = 0

   7 mod 3 = 1

   (-6) mod 3 = 0

   (-7) mod 3 = -1

   (-5) mod 3 = -2

   (-15) mod (-7) = -1

En las operaciones aritméticas, debe asegurarse que el resultado de sumar, restar o multiplicar dos valores, no produzca un resultado fuera de los rangos definidos por la implementación para los diferentes tipos.

OPERADORES RELACIONALES: Una RELACIÓN consiste de dos operandos separados por un operador relacional. Si la relación es satisfecha, el resultado tendrá un valor booleano true ; si la relación no se satisface, el resultado tendrá un

valor false. Los operadores deben ser del mismo tipo, aunque los valores de tipo real, integer y byte pueden combinarse como operandos en las relaciones.

A continuación se describen los operadores relacionales utilizados en Pascal:

Símbolo	Significado
=	IGUAL que
<>	NO IGUAL que
<	MENOR que
>	MAYOR que
<=	MENOR o IGUAL que
>=	MAYOR o IGUAL que

Tabla 2.4 Operadores relacionales de Pascal

Ejemplos:

Relación	Resultado
20 = 11	false
15 < 20	true
PI > 3.14	true
'A' < 20	false
'A' = 65	true

Tabla 3.5 Ejemplos de operadores relacionales de Pascal

OPERADORES LÓGICOS: Al igual que las relaciones, en las operaciones con operadores lógicos se tienen resultados cuyo valor de verdad toma uno de los valores booleanos true o false.

Los operadores lógicos en Pascal son:

NOT Sintaxis : not operando Descripción: Invierte el valor de verdad de operando. Ejemplo: : Si bandera tiene un valor de verdad true, not bandera produce un resultado con valor de verdad false.
AND Sintaxis : operando.1 and operando.2 Descripción : Produce un resultado con valor de verdad true cuando ambos operandos tienen valor de verdad true; en cualquier otro caso el resultado tendrá un valor de verdad false.
OR Sintaxis : operando.1 or operando.2 Descripción : Produce un resultado con valor de verdad false cuando ambos operadores tienen valores de verdad false; en cualquier otro caso el resultado tendrá un valor de verdad true.
XOR Sintaxis : operando.1 xor operando.2 Descripción : Un operando debe tener valor de verdad true y el otro false para que el resultado tenga valor de verdad true.

Tabla 2.6 Operadores lógicos en Pascal

Turbo Pascal también permite las siguientes operaciones entre los bits de operandos exclusivamente de tipo entero:

AND Sintaxis : operando.1 and operando.2 Descripción: Pone a ceros los bits de operando.2 cuyos correspondientes en operando.1 estén en ceros. Los valores se pasan a binario, y, sobre cada bit de operando.1 se realiza la operación and lógica con el correspondiente bit de operando.2. Ejemplo : 29 and 30 = 28 Cuya forma en binario es : 0000000000011101 = 29 (operando.1) and 0000000000011110 = 30 (operando.2) _____ 0000000000011100 = 28 (resultado)
OR (o inclusiva) Sintaxis : operando.1 or operando.2 Descripción : Pone a uno los bits de operando.1 cuyos correspondientes bits en operando.2 están a uno. Ejemplo : 17 or 30 = 31 En binario: 0000000000010001 = 17 (operando.1) or 0000000000011110 = 30 (operando.2) _____ 0000000000011111 = 31 (resultado)
XOR (o exclusiva) Sintaxis : operando.1 xor operando.2 Descripción : Invierte el estado de los bits de operando.1, cuyos correspondientes en operando.2 están a uno. Ejemplo : 103 xor 25 = 126 En binario: 0000000001100111 = 103 (operando.1) xor 0000000000011001 = 25 (operando.2) _____ 0000000001111110 = 126 (resultado)
SHL Sintaxis : operando.1 shl operando.2 Descripción : Desplaza hacia la izquierda los bits de operando.1, el número de posiciones establecidas por operando.2. Los bits que salen por el extremo izquierdo se pierden.

Ejemplo : 10 shl 2 = 40 En binario: 0000000000001010 = 10 (operando.1) shl 2 <= 0000000000101000 = 40 (resultado) (operando.2)
SHR Sintaxis : operando.1 shr operando.2 Descripción : Desplaza hacia la derecha los bits de operando.1 el número de posiciones establecidas por operando.2. Los bits que salen por el extremo derecho se pierden Ejemplo : 125 shr 3 = 15 En binario : 0000000001111101 = 125 (operando.1) shr 3 => 0000000000001111 = 15 (resultado) (operando.2)

Tabla 2.7 Operaciones entre los bits de operandos exclusivamente de tipo entero.

FUNCIONES ESTANDARES.

Las funciones estandares también llamadas intrinsecas, están implementadas en el lenguaje. El programador no necesita elaborarlas , sino simplemente hacer uso de ellas.

Algunas funciones estandares son las siguientes:

Función	Operación	Tipo de operando	Tipo de resultado
ABS (X)	Valor absoluto de X	Entero Real	Entero Real
ARCTAN (X)	Ángulo en radianes, cuya tangente es X.	Entero	Real
CHR (X)	Da el carácter correspondiente a la posición en el código ASCII.	Entero	Carácter
COS (X)	Coseno de X radianes.	Entero Real	Real
EOF (A)	Indica si el apuntador del archivo A se encuentra en el final del archivo INPUT.	Archivo	Booleano
EOLN (A)	Indica si el apuntador del archivo A se encuentra en el final de una línea o registro de este. Se usa EOLN si se prefiere el archivo INPUT.	Archivo	Booleano
Ln (X)	Valor del numero elevado a la	Entero	Real

	potencia de X		
ODD (X)	Indica si X es impar	Entero	Booleano
ORD (X)	Da la posición	Carácter	Entero enumerado

2.3.3 Programación estructurada con Pascal.

Tipos estructurados

En Pascal, se pueden definir, a partir de los datos simples, otros tipos más complejos conocidos como tipos estructurados.

Cuando se declara una variable como de un tipo estructurado, se puede manipular la estructura completa, o bien trabajar con los datos individuales que la forman.

A continuación, se describirán los diferentes tipos estructurados que pueden manejarse en Pascal.

Cadenas (String)
Arreglos (Array)
Registros (Record)
Conjuntos (Set)
Archivos

Tabla 2.8 Tipos estructurados que pueden manejarse en Pascal

CADENAS (STRINGS): Turbo Pascal proporciona el tipo string para el procesamiento de cadenas (secuencias de caracteres).

La definición de un tipo string debe especificar el número máximo de caracteres que puede contener, esto es, la máxima longitud para las cadenas de ese tipo. La longitud se especifica por una constante entera en el rango de 1 a 255.

El formato para definir un tipo string es:

```
<identificador> = string [limite_superior];
```

Tabla 2.9 Formato para definir un tipo string.

Las variables de cadena se declaran en la sección Var o Type.

Declaración en Var:

```
Var
nombre    : string[30];
domicilio : string[30];
ciudad    : string[40];
```

Declaración en Type:

```
Type
cad30     : string[30];
cad40     : string[40];
Var
nombre    : cad30;
domicilio : cad30;
ciudad    : cad40;
```

Una Vez declaradas las variables se pueden realizar asignaciones u operaciones de lectura/escritura.

```
nombre    := 'Egrid Lorely Castro Gonzalez' ;
domicilio := 'Altamirano #220';
ciudad    := 'La Paz B.C.S.';
```

Figura 2.2 Asignaciones u operaciones de lectura/escritura.

El contenido de la cadena se debe encerrar entre apóstrofes. Si se desea que figure un apóstrofe en una cadena, es preciso doblarlo en la cadena. Los procedimientos de Entrada/Salida son de la siguiente forma:

```
ReadLn (nombre);
WriteLn('Hola ',nombre);
```

Figura 2.3 Procedimientos de Entrada/Salida

Las variables de tipo cadena pueden ocupar la máxima longitud definida, más un octeto que contiene la longitud actual de la variable. Los caracteres que forman la cadena son numerados desde 1 hasta la longitud de la cadena.

Ejemplo:

```
Var
  nombre : string[10];
begin
  nombre := 'Susana';
end.
```

Obsérvese que el primer byte no es el carácter '6' si no el número 6 en binario (0000 0110) y los últimos bytes de la cadena hasta 10 (7-10) contienen datos aleatorios.

Una cadena en Turbo Pascal tiene dos longitudes:

1. Longitud física : Es la cantidad de memoria que ocupa realmente, está se establece en tiempo de compilación y nunca cambia
2. Longitud lógica: Es el número de caracteres almacenados actualmente en la variable cadena. Este dato puede cambiar durante la ejecución del programa.

Es posible acceder a posiciones individuales dentro de una variable cadena, mediante la utilización de corchetes que dentro de ellos se especifica el número índice dentro de la cadena a utilizar así para el ejemplo anterior se tiene :

```
nombre[1] ==> 'S'
nombre[2] ==> 'u'
nombre[3] ==> 's'
nombre[4] ==> 'a'
nombre[5] ==> 'n'
nombre[6] ==> 'a'
```

Operaciones entre cadenas: Las operaciones básicas entre cadenas son : asignación, comparación y concatenación. Es posible asignar una cadena a otra cadena, incluso aunque sea de longitud física más pequeña en cuyo caso ocurriría un truncamiento de la cadena.

Ejemplo:

```
Var
  nombre : String[21];
    .
    .
    .
  nombre := 'Instituto Tecnológico de La Paz';
```

El resultado de la asignación en la variable nombre será la cadena 'Instituto Tecnológico'.

Las comparaciones de las cadenas de caracteres se hacen según el orden de los caracteres en el código ASCII y con los operadores de relación.

'0' < '1' '2' > '1' 'A' < 'B' 'm' > 'l'

Reglas de comparación de cadenas: Las dos cadenas se comparan de izquierda a derecha hasta que se encuentran dos caracteres diferentes. El orden de las dos cadenas es el que corresponde al orden de los dos caracteres diferentes. Si las dos cadenas son iguales pero una de ella es más corta que la otra, entonces la más corta es menor que la más larga.

Ejemplo:

'Alex' > 'Alas'
{puesto que 'e' > 'a'}
'ADAN' < 'adan'
{puesto que 'A' < 'a'}
'Damian' < 'Damiana'
{'Damian' tiene menos caracteres que 'Damiana'}
'El gato' < 'Los gatos'
{puesto que (blanco) < 's'}

Otra operación básica es la concatenación. La concatenación es un proceso de combinar dos o más cadenas en una sola cadena. El signo + se puede usar para concatenar cadenas (al igual que la función concat), debiendo cuidarse que la longitud del resultado no sea mayor que 255.

Ejemplos:

'INSTITUTO '+'TECNOLOGICO'='INSTITUTO TECNOLOGICO'
'CONTAB'+'.'+'PAS'= 'CONTAB.PAS'

Se puede asignar el valor de una expresión de cadena a una variable cadena, por ejemplo:

fecha := 'lunes';
y utilizar la variable fecha en :
frase:='El próximo '+fecha+' inician las clases';

Si la longitud máxima de una cadena es excedida, se pierden los caracteres sobrantes a la derecha. Por ejemplo, si fecha hubiera sido declarada del tipo string[7], después de la asignación contendría los siete primeros caracteres de la izquierda (CENTENA).

ARREGLOS (ARRAY): Un arreglo está formado por un número fijo de elementos contiguos de un mismo tipo. Al tipo se le llama tipo base del arreglo. Los datos individuales se llaman elementos del arreglo.

Para definir un tipo estructurado arreglo, se debe especificar el tipo base y el número de elementos.

Un array se caracteriza por:

1. Almacenar los elementos del array en posiciones de memoria continua
2. Tener un único nombre de variable que representa a todos los elementos, y éstos a su vez se diferencian por un índice o subíndice.
3. Acceso directo o aleatorio a los elemento s individuales del array.

Los arrays se clasifican en:

- Unidimensionales (vectores o listas)
- Multidimensionales (tablas o matrices)

El formato para definir un tipo array es:

nombre_array = array [tipo subíndice] of tipo

nombre_array identificador válido tipo subíndice puede ser de tipo ordinal: boolean o char, un tipo enumerado o un tipo subrango. Existe un elemento por cada valor del tipo subíndice tipo describe el tipo de cada elemento del vector; todos los elementos de un vector son del mismo tipo

Tabla 2.10 Formato para definir un tipo array

Las variables de tipo array se declaran en la sección Var o Type.

Declaración en Var:

```
Var
  nombres   : array[1..30] of string[30];
  calif     : array[1..30] of real;
  numero    : array[0..100] of 1..100;
```

Declaración en Type:

```
Type
  nombres   : array[1..30] of string[30];
  calif     : array[1..30] of real;
  numero    : array[0..100] of 1..100;
Var
  nom       : nombres;
  califica  : calif;
  num       : numero;
```

REGISTROS (RECORD): Un registro es una estructura que consiste de un número fijo de componentes llamados campos. Los campos pueden ser de diferentes tipos y deben tener un identificador de campo.

La definición de un tipo registro debe consistir de la palabra reservada record, seguida de una lista de campos y terminada por el identificador reservado end.

```
type
  tipo_reg = record
               lista id1:tipo 1;
               lista id2:tipo 2;
                     .
                     .
                     .
               lista idn:tipo n
             end;
```

Figura 2.4 Definición de un tipo registro

tipo_reg: Nombre de la estructura o dato registro.
lista id: Lista de uno o más nombres de campos separados por comas.
tipo: Puede ser cualquier tipo de dato estándar o definido por el usuario

CONJUNTOS (SETS): Un conjunto es una colección de objetos relacionados. Cada objeto en un conjunto es llamado miembro o elemento del conjunto.

Aunque en matemáticas no hay restricciones para que los objetos puedan ser elementos de un conjunto, Pascal sólo ofrece una forma restringida de conjuntos, por lo que:

1. Los elementos de un conjunto deben ser del mismo tipo, llamado el tipo base.
2. El tipo base debe ser un tipo simple, excepto el tipo real.

Representación de conjuntos:

Elementos	Notación Matemática	Pascal
1,2,3,4,5	{1,2,3,4,5}	[1,2,3,4,5]
a,b,c	{a,b,c}	['a','b','c']

Tabla 32.11 Representación de conjuntos

Aunque se puede utilizar notación de tipo subrango para especificar secuencia de valores que pertenezcan a un conjunto, los elementos del conjunto no tienen una ordenación interna particular. La única relación entre los miembros de un conjunto es: existe o no existe en el conjunto.
[5,5] y [5] son equivalentes (contienen un sólo elemento)

ARCHIVOS (FILE): Un tipo archivo se define con los identificadores reservados FILE OF, seguidas por el tipo de los componentes del archivo.

Por ejemplo:

```
Type
  empleado = file of Record
               nombre:string[30];
               sueldo:real;
             end;
Var
  nomina : empleado ;
```

También puede escribirse así:

```
Type
h  empleado = Record
              nombre:string [30];
              sueldo:real;
             end;
Var
```

nomina : file of empleado;

ESTRUCTURAS DE CONTROL

Se denominan estructuras de control a aquellas que determinan qué instrucciones deben ejecutarse y qué números de veces. Existen dos tipos de estructuras de control:

- Alternativas o de selección.
- Repetitivas o de iteración.

Estructuras alternativas. Son aquellas que bifurcan o digieren la ejecución de un programa hacia un grupo de sentencias u otros dependiendo del resultado de una condición. Las dos sentencias alternativas de Pascal son:

- Sentencia alternativa simple IF-THEN-ELSE
- Sentencia alternativa múltiple CASE-OF.

SENTENCIA IF-THEN-ELSE

La sintaxis de esta sentencia es la siguiente:

```
IF (expresión lógica o booleana)
THEN
     Sentencia 1 (simple o compuesta)
ELSE
     Sentencia 2 (simple o compuesta);
```

El diagrama de flujo de la sentencia IF se representa en la Figura 2.5:

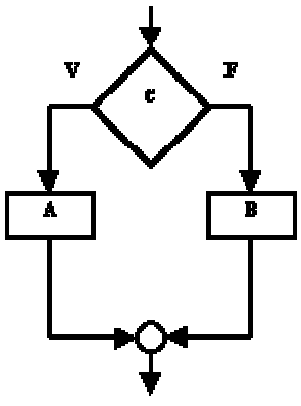


Figura 2.5 Diagrama de Flujo Sentencia IF.

Ejemplo:

```
IF n>0 then writeln (‘Numero positivo’);
IF n>0 then
        Writeln (‘numero positivo’)
ELSE
        Writeln (‘Numero negativo’);
```

No puede existir un punto y coma inmediatamente antes de una palabra ELSE ya que sería interpretado como final de IF.

Ejemplo del uso de IF:

```
Program Colores;
Var
    Color : string;
Begin
     Writeln (‘¿Cuál es tu color preferido?’)
     Readln (Color);
IF color = rojo THEN
     Writeln (‘Tu color representa: Amor’);
IF color = verde THEN
     Writeln (‘Tu color representa: Vida’);
IF color = azul THEN
     Writeln (‘Tu color representan: Seguridad’);
IF color = amarillo THEN
    Writeln (‘Tu color representa: Esperanza’);
IF color = naranja THEN
    Writeln (‘Tu color representa: Energia’);
END.
```

SENTENCIA CASE OF

La sintaxis de esta sentencia es la siguiente:

```
CASE (expresión o variable) OF
     (lista de constante 1) : (sentencia 1);
     (lista de constante 2) : (sentencia 2);
     (lista de constante 3) : (sentencia 3);
…
     (lista de constante N) : (sentencia N);
ELSE (sentencia)
…
END;
```

Ejemplo:

```
Program Menu;
Var
   Numerodia: integer;

Begin
Write (‘Introduzca el ordinal de un día laborable de la semana:’);
Readln (numerodia);
Write (‘Hoy es:’);

CASE numerodia OF
         1 : Writeln (‘Lunes’);
         2 : Writeln (‘Martes’);
         3 : Writeln (‘Miercoles’);
         4 : Writeln (‘Jueves’);
         5 : Writeln (‘Viernes’);
         6 : Writeln (‘Sabado’);
ELSE Writeln (‘¡¡¡¡Domingo!!!! No es día laborable’);
END;
```

El diagrama de flujo de una sentencia CASE se representa en la Figura 2.6.

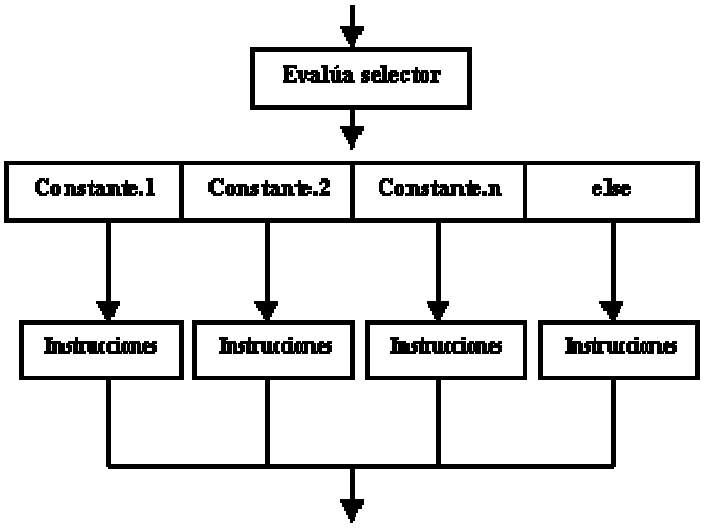


Figura 2.6 Diagrama de Flujo Sentencia CASE.

LAS ESTRUCTURAS REPETITIVAS. Son aquellas que crean un bucle (repetición continua de un conjunto de instrucciones) en la ejecución de un programa respecto de un grupo de sentencias en función de una condición. Las tres sentencias repetitivas de Turbo Pascal son:

- SENTENCIA WHILE
- SENTENCIA REPEAT-UNTIL
- SENTENCIA FOR

SENTENCIA WHILE

Indica al ordenador que se ejecuten una o más sentencias mientras se cumpla una determinada condición establecida por una variable o expresión booleana.
Esta sentencia comprueba inicialmente si la condición es verdadera. Si la condición es verdadera se ejecutan las sentencias mientras la condición de su enunciado sea verdadera y finaliza cuando la condición es falsa.
Dado que la condición puede ser falsa inicialmente, es decir antes de comenzar el bucle, habrá casos en el que el bucle no se ejecute.

La sintaxis es la siguiente:

```
WHILE condición DO
BEGIN
(sentencia 1);
…
(sentencia N);
END;

WHILE condición DO
(sentencia);
```

Las características del Bucle While se ejecuta mientras la condición sea verdadera, y dentro de bucle debe existir, por lo menos, una sentencia que modifique el valor de la variable o expresión, de lo contrario se puede producir una situación de bucle infinito. Si la expresión lógica es falsa al comenzar el bucle, este no se realizará.

El diagrama de flujo de la sentencia while es la siguiente:

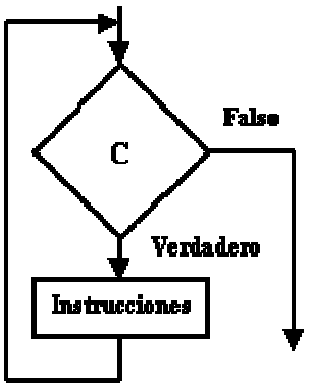


Figura 2.7 Diagrama de Flujo Sentencia WHILE.

Ejemplo:

Escribir los N primeros números naturales, donde N es un valor introducido por el usuario.

```
Program escribeenteros;
Var N, contador:integer;
Begin
     Write (‘Introduzca numero máximo de enteros: ‘);
     Readln (N);  Contador:=1;
     While contador <=N do
       Begin
          While (contador:5);     Contador:=contador+1;
```

```
        End;
    Writeln ('Fin de programa. Contador = ',contador);
End.
```

SENTENCIA REPEAT UNTIL

Ejecuta las sentencias comprendidas entre las palabras reservadas REPEAT y UNTIL hasta que la expresión o variable sea verdadera.

La sintaxis es la siguiente:

```
REPEAT
        Begin
      (Sentencia);
      (Sentencia);
       End;

UNTIL condición;
```

Algunas de las características que tiene el Bucle Repeat, es que se ejecutan siempre una vez, por lo menos, y la terminación del bucle se produce cuando el valor de la expresión lógica o condición de salida es verdadera. Se ejecuta hasta que la expresión es verdadera, es decir, se ejecuta mientras la expresión sea falsa.

El diagrama de la sentencia Repeat se representa en la Figura 2.8.

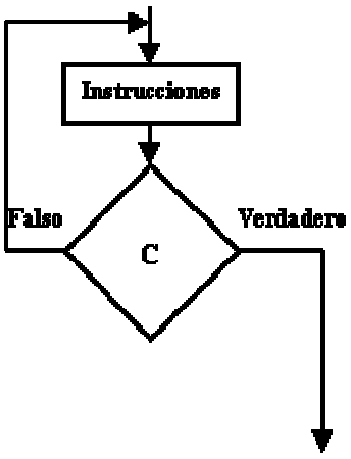


Figura 2.8 Diagrama de Flujo Sentencia REPEAT.

Ejemplo:

```
Program escribeenteros;
Var N, contador:integer;
Begin
      Write (‘Introduzca numero máximo de enteros: ‘);
      Readln (N);  Contador:=0;
      Repeat
           Contador:=contador+1;
            Write (contador:5)
      Until contador= N;
       Writeln (‘Fin del programa. Contador=’,contador)
End.
```

SENTENCIA FOR

La sentencia For repite la ejecución de una o varias sentencias un numero fijo de veces, previamente establecido. Necesita una variable de control del bucle que es necesariamente de tipo ordinal, ya que el bucle se ejecuta mientras la variable de control toma una serie consecutiva de valores de tipo ordinal, comprendidos entre dos valores extremos (interior y superior).

- Formato ascendente:

```
FOR variablecontrol:=valorinicial TO valorfinal DO
       (sentencia);
```

- Formato descendente:

```
FOR variablecontrol:= valorinicial DOWNTO valorfinal DO
      (sentencia);
```

Donde (sentencia) puede ser una sentencia simple o compuesta.

Algunas de las características de el Bucle For es que aunque a primera vista pueda resulta más atractivo FOR, existen limitaciones en su aplicación ya que el bucle FOR siempre se incrementa o decrementa (de uno en uno) los valores de la variable de control de bucle y no de dos en dos o de tres en tres, o con valores fraccionarios. El número de iteraciones de un bucle FOR siempre es fijo y se conoce de antemano:

Valor final – Valor inicial + 1

Ejemplo:

```
Program escribeeneteros;
Var N, contador: integer;
   Begin
       Write (‘Introduzca numero máximo de enteros: ‘);
       Readln (N);
       For contador:= 1 to n do
             Write (contador:5);
        Writeln
End.
```

Por lo tanto es necesario saber cuándo utilizar las diferentes sentencias While/Repeat/For. Así que utilizar las sentencias o estructuras FOR cuando se conozca en número de iteraciones, y siempre que la variable de control del bucle sea de tipo ordinal. Utilizar la estructura REPEART-UNTIL cuando el bucle se realice por lo menos una vez. En todos los demás casos utilizar la sentencia WHILE.

2.3.4 Tipos de datos definidos por el usuario.

Definición de tipos

Además de identificadores, los datos deben tener asignado algún tipo que indique el espacio de memoria en que se almacenarán y que al mismo tiempo evita el error de tratar de guardar un dato en un espacio insuficiente de memoria.

Un tipo de dato en Pascal puede ser cualquiera de los tipos predefinidos (integer, real, byte, boolean, char), o algún otro definido por el programador en la parte de definición de tipos .

Los tipos definidos por el programador deben basarse en los tipos estándar predefinidos, para lo cual, debe iniciar con el identificador reservado Type seguido de una o más asignaciones de tipo separadas por punto y coma. Cada asignación de tipo debe consistir de un identificador de tipo, seguido por un signo de igual y un identificador de tipo previamente definido.

La asignación de tipos a los datos tiene dos objetivos principales:

1. Detectar errores de operaciones en programas.
2. Determinar cómo ejecutar las operaciones.

Pascal se conoce como un lenguaje "fuertemente tipeado" (strongly-typed) o de tipos fuertes. Esto significa que todos los datos utilizados deben tener sus tipos declarados explícitamente y el lenguaje limita la mezcla de tipos en las expresiones. Pascal detecta muchos errores de programación antes de que el programa se ejecute.

Los tipos definidos por el programador pueden utilizarse para definir nuevos tipos, por ejemplo:

```
Type
  entero      = integer;
  otro_entero = entero;
```

2.3.5 Manejo de archivos.

Un archivo es el módulo básico de información manejado por el Sistema Operativo. El Sistema Operativo es un conjunto de programas cuya función es administrar los recursos del Sistema de Cómputo. Por ejemplo, un programa fuente es almacenado como un archivo. Primero es introducido en la memoria RAM por medio de un programa editor, y después es almacenado como un archivo de texto en un medio de almacenamiento permanente (cinta, disco flexible, disco duro). Una vez que el programa fuente ha sido compilado, el programa resultante de la compilación viene a ser un archivo binario.

En Pascal, un archivo es una secuencia de elementos que pertenecen al mismo tipo o estructura, esto es que un archivo puede ser una secuencia de caracteres, números o registros, por lo que su representación lógica puede hacerse como una secuencia de módulos del mismo tamaño.

En el vocabulario de manejo de archivos, a cada elemento de un archivo se le llama registro. En Pascal, la numeración de los registros empieza con el número CERO, por lo que al elemento_1 se le llamará registro 0, al elemento_2 registro 1, y así sucesivamente hasta llegar a la marca de fin de archivo EOF.

Turbo Pascal difiere significativamente de Pascal estándar por la forma en que maneja los archivos.

En Pascal estándar, los archivos son formalmente definidos independientemente del medio en que residan. Este método de definición fue inspirado por los archivos de tarjetas perforadas y cintas magnéticas, las cuales eran los medios de almacenamiento comúnmente usados cuando Pascal fue definido por primera vez. Como resultado, todo acceso a cualquier archivo en Pascal estándar es

secuencial (registro por registro) tal como se realiza en las tarjetas perforadas y cintas magnéticas.

En Turbo Pascal los archivos son definidos como archivos de disco. Los discos son actualmente los dispositivos de almacenamiento más utilizados en las microcomputadoras. Los mecanismos de acceso secuencial proporcionados por Pascal estándar son algunas veces inconvenientes e insuficientes para los archivos basados en discos de acceso aleatorio, por lo que Turbo Pascal provee nuevas estructuras y mecanismos de acceso a los archivos.

La primera gran diferencia entre Turbo Pascal y Pascal estándar, es la forma en que enlazan los archivos a un programa. En Pascal estándar, se abren los archivos referenciando su nombre de archivo en el encabezado del programa, y se cierran cuando el programa termina. En Turbo Pascal, los archivos de disco deben enlazarse a una variable de archivo particular con el procedimiento:

Assign(variable_archivo,nombre_archivo);

Y deben ser preparados para procesarse (abiertos) con: reset(variable_archivo) o rewrite(variable_archivo) antes de ser utilizados.

Además, los archivos deben ser explícitamente cerrados por medio de close(variable_archivo), después de que han sido utilizados, o se perderán los datos que estén en la memoria auxiliar (variable_archivo) .

variable_archivo es el nombre de la memoria auxiliar (buffer), por medio de la cual el programa manejará los datos hacia o desde el archivo en disco.
nombre_archivo es el nombre que identifica al archivo en el disco.

reset abre un archivo existente para procesamiento y coloca el apuntador de registro en el primer registro (0).
rewrite crea un nuevo archivo (o sobre-escribe en uno existente) y lo abre para procesamiento con el apuntador de registro colocado en el registro 0.

En el caso de un archivo de tipo text, reset hará que el archivo sólo pueda ser usado para operaciones de lectura , mientras que rewrite sólo permitirá operaciones de escritura.
Los nombres de archivo válidos son cadenas de 1 a 8 caracteres seguidos por una extensión opcional consistente de un punto y hasta tres caracteres. A estos nombres también se les llama "nombres de archivo externo", puesto que son los nombres con que se graban en el disco.

Tipos de archivos

Existen tres tipos de archivos de datos en Turbo Pascal:

1. texto (text) o secuenciales (acceso secuencial),
2. tipeados (tipificados) o con tipo (file of) (acceso aleatorio), aleatorios,
3. no tipeados (no tipificados) o sin tipo (file).

Archivos de texto (secuenciales) :	Son archivos que contienen texto (carácter ASCII)
Archivos con tipo (aleatorios) :	Archivos que contienen datos de cualquier tipo como integer, real, byte, record, datos con estructuras.
Archivos sin tipo :	Archivos en los que no se conoce su estructura ni su contenido; están concebidos para acceso de bajo nivel a los datos de un disco (E/S de bytes).

Tabla 2.12 Tipos de archivos de datos en Turbo Pascal

Tipos de acceso a un archivo

Existen dos modalidades para acceder a un archivo de datos : acceso secuencial y acceso directo o aleatorio. El acceso secuencial exige elemento a elemento, es necesaria una exploración secuencial comenzando desde el primer elemento. El acceso directo permite procesar o acceder a un elemento determinado haciendo una referencia directamente por su posición en el soporte de almacenamiento. Pascal estándar sólo admite el acceso secuencial, pero Turbo Pascal permite el acceso directo.

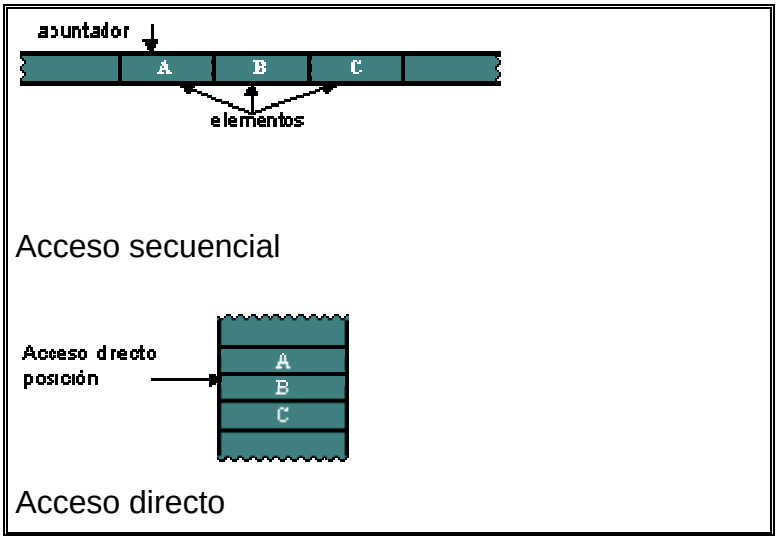


Figura 2.9 Tipos de acceso en Pascal estándar y Turbo Pascal.

La declaración de un archivo consta de dos pasos:

1. Declaración del tipo de archivo adecuado:
1.1 file of char archivo de texto
file of text
1.2 file of <tipo> archivo con tipo
1.3 file archivo sin tipo
2. Declaración de una variable archivo de un tipo de archivo declarado.

Declaración de un tipo archivo (file)

Un tipo archivo se declara de igual modo que cualquier otro tipo de dato definido por el usuario: en la sección de declaración de tipos (type).

```
Type
  nombre = file of tipo de datos
```

Figura 2.10 Declaración de un Archivo

nombre : Identificador que se asigna como nombre del tipo archivo.
tipo de datos: Tipo de datos de los elementos del archivo

Ejemplos:

```
type ArchEntero = file of integer;
{archivo de enteros}
type ArchCarac  = file of char;
{archivo de carácteres}
type nomina     = file of empleado;
{archivo de registros}
type ciudad     = file of string[20];
{archivo de cadenas}
```

Variable tipo archivo (file):
Para definir un archivo con tipos, simplemente declare una variable archivo.

Ejemplo:

```
Var
  arch_nomina : nomina;
  enteros     : ArchEntero;
```

O también

```
Var
  arch_nomina : file of empleado;
  enteros     : file of integer;
```

Variables de tipo texto y sin tipo:

Este tipo de variables no requiere ninguna declaración de tipos; así pues, se puede declarar con un identificador predefinido (Text,File):

```
Var
   texto   : text;
   Archivo : file;
```

2.3.6 Ejemplos prácticos

El Pascal ofrece muchas más posibilidades, como por ejemplo la definición de tipos por el usuario, o el tipo Set (conjunto), de gran interés dado que no aparecen en la mayoría de los lenguajes. Otro tipo a destacar es el denominado puntero. La recursividad es una característica de mucha utilidad, es decir que cierta instrucción se puede llamar a si misma. Aquí no se tratan estas posibilidades aunque se usan ampliamente en la programación a escala avanzada en Pascal.

2.4 Aplicaciones Generales

PASCAL es un lenguaje de programación de alto nivel de propósito general; esto es, se puede utilizar para escribir programas para fines científicos y comerciales. Fue diseñado por el profesor Niklaus (Nicolás) Wirth en Zurich, Suiza, al final de los años 1960 y principios de los 70's. Wirth diseñó este lenguaje para que fuese un buen lenguaje de programación para personas comenzando a aprender a programar. Pascal tiene un número relativamente pequeño de conceptos para aprender a denominar. Su diseño facilita escribir programas usando un estilo que esta generalmente aceptado como práctica estándar de programación buena. Otra de las metas del diseño de Wirth era la implementación fácil.

El principio de Pascal y sus aplicaciones

La presión aplicada en un punto de un líquido contenido en un recipiente se transmite con el mismo valor a cada una de las partes del mismo.
Este enunciado, obtenido a partir de observaciones y experimentos por el físico y matemático francés Blas Pascal (1623-1662), se conoce como principio de Pascal.

El principio de Pascal puede ser interpretado como una consecuencia de la ecuación fundamental de la hidrostática y del carácter incompresible de los líquidos. En esta clase de fluidos la densidad es constante, de modo que de acuerdo con la ecuación p = po + · g · h si se aumenta la presión en la superficie libre, por ejemplo, la presión en el fondo ha de aumentar en la misma medida, ya que · g · h no varía al no hacerlo h.

La prensa hidráulica constituye la aplicación fundamental del principio de Pascal y también un dispositivo que permite entender mejor su significado. Consiste, en esencia, en dos cilindros de diferente sección comunicados entre sí, y cuyo interior está completamente lleno de un líquido que puede ser agua o aceite. Dos émbolos de secciones diferentes se ajustan, respectivamente, en cada uno de los dos cilindros, de modo que estén en contacto con el líquido. Cuando sobre el émbolo de menor sección S1 se ejerce una fuerza F1 la presión p1 que se origina en el líquido en contacto con él se transmite íntegramente y de forma instantánea a todo el resto del líquido; por tanto, será igual a la presión p2 que ejerce el líquido sobre el émbolo de mayor sección S2, es decir:

p1 = p2

con lo que:

y por tanto:

Si la sección S2 es veinte veces mayor que la S1, la fuerza F1 aplicada sobre el émbolo pequeño se ve multiplicada por veinte en el émbolo grande.
La prensa hidráulica es una máquina simple semejante a la palanca de Arquímedes, que permite amplificar la intensidad de las fuerzas y constituye el fundamento de elevadores, prensas, frenos y muchos otros dispositivos hidráulicos de maquinaria industrial.

2.5 Conclusiones

En conclusión podemos decir que el Leguaje de programación Pascal es lo suficientemente poderoso para programar y crear gran parte de los algoritmos de la vida cotidiana, pero como veremos existen más Lenguajes de programación y más poderosos o inclusive más faciles de utilizar.

CAPÍTULO 3

3. HERRAMIENTA VISUAL DELPHI

3.1 Objetivo

Proporcionar los conocimientos básicos teóricos y prácticos para el diseño y construcción de aplicaciones con interfaz gráfica de usuario, utilizando programación orientada a eventos.

3.2 Introducción

Delphi es una de las herramientas de desarrollo rápido de aplicaciones más potentes y difundidas a nivel mundial, pero es también un entorno de programación completo en el que hay muchos elementos.

Delphi se divide en tres secciones, el compilador (con su "encadenador"), la librería, y el IDE (Ambiente de desarrollo integrado, o Integrated Development Environment). El compilador/encadenador es un programa que crea el archivo ejecutable de Windows (PE) estilo Intel, sin ningún interprete de por medio. La librería es código que nos permite usar todas las capacidades de Delphi. La librería esta escrita en su totalidad en Object Pascal (es una librería "de clases" estilo MFC, llamada VCL), y esta totalmente orientada a objetos.

Una nota interesante es que el IDE esta hecho en Delphi, y utiliza las mismas librerías para compilar su programa. Esto quiere decir dos cosas:
Primero, que todo lo que puede hacer el IDE es posible (lo cual es notorio contraste con lenguajes como Visual Basic).
Segundo, que el IDE esta abierto, lo cual quiere decir que se puede no solo extender la librería para que Delphi utilice los "componentes" diseñados, sino que además puede extender el IDE para hacer "expertos" y ayudas de programación.

El "programa principal" de Delphi es un archivo de texto ASCII con extensión .DPR. Ésta extensión quiere decir Delphi PRoject (proyecto de Delphi).

Para cada una de las ventanas que usted diseña en el IDE, Delphi crea una "unidad". Una unidad es un archivo individual (también de texto ASCII) que representa en general a un objeto, o a una agrupación lógica de funciones. En el caso de los "objetos" que son formas, Delphi también crea un archivo "DFM" (Delphi Form) para guardar la apariencia del diseño de las mismas (las formas son simplemente archivos de recursos en un formato especial que solo funciona en Delphi).

La Figura 3.1 muestra el archivo DPR que se crea cuando se comienza a trabajar con Delphi. En general se puede crear programas muy complejos sin jamás modificar el archivo DPR (también llamado archivo de proyecto). Pero es importante saber como funciona. El archivo de Proyecto "Project1.Dpr" tiene la siguiente apariencia:

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Figura 3.1 Muestra el archivo DPR que se crea cuando se comienza a trabajar con Delphi.

Una vez que Delphi ha especificado todo lo que va a usar el proyecto, el programa inicializa la aplicación (Application.Initialize), crea la forma Form1 a partir de la definición de objeto "TForm1" usando "CreateForm", y la aplicación "corre" (Application.Run). Cualquier forma creada con CreateForm está "viva" hasta que el Application.Run termina. Después de esto (normalmente cuando "Run" recibe el mensaje "Terminate") el programa termina.

Las nuevas versiones de Delphi, incorporan soporte para el desarrollo de multiplataformas con la nueva Biblioteca de componentes para multiplataformas

(CLX), el nuevo motor para base de datos dbExpress, servicios Web, más ampliaciones Web, más ampliaciones IDE y multitud de componentes, etc.

La última versión de Delphi es el primer paso para los desarrolladores en la migración de Win32 a la nueva plataforma. Delphi, al igual que sus predecesores, es un entorno de desarrollo Windows de 32 bits con nuevas características y la tecnología específica para ayudar a los desarrolladores Delphi a entrar en el mundo .NET.

3.3 Desarrollo

El entorno de programación típico consta en Delphi de cuatro ventanas que comparten el espacio disponible de la pantalla. Cada una de estas ventanas puede modificarse, cerrarse y volverse a abrir mediante el menú View.

3.3.1 Interfaces gráficas de usuario.

Estas ventanas fundamentales son:

- La ventana principal
- El inspector de objetos (object inspector)
- El editor de código fuente
- La ventana (o ventanas) de programa (Forms)

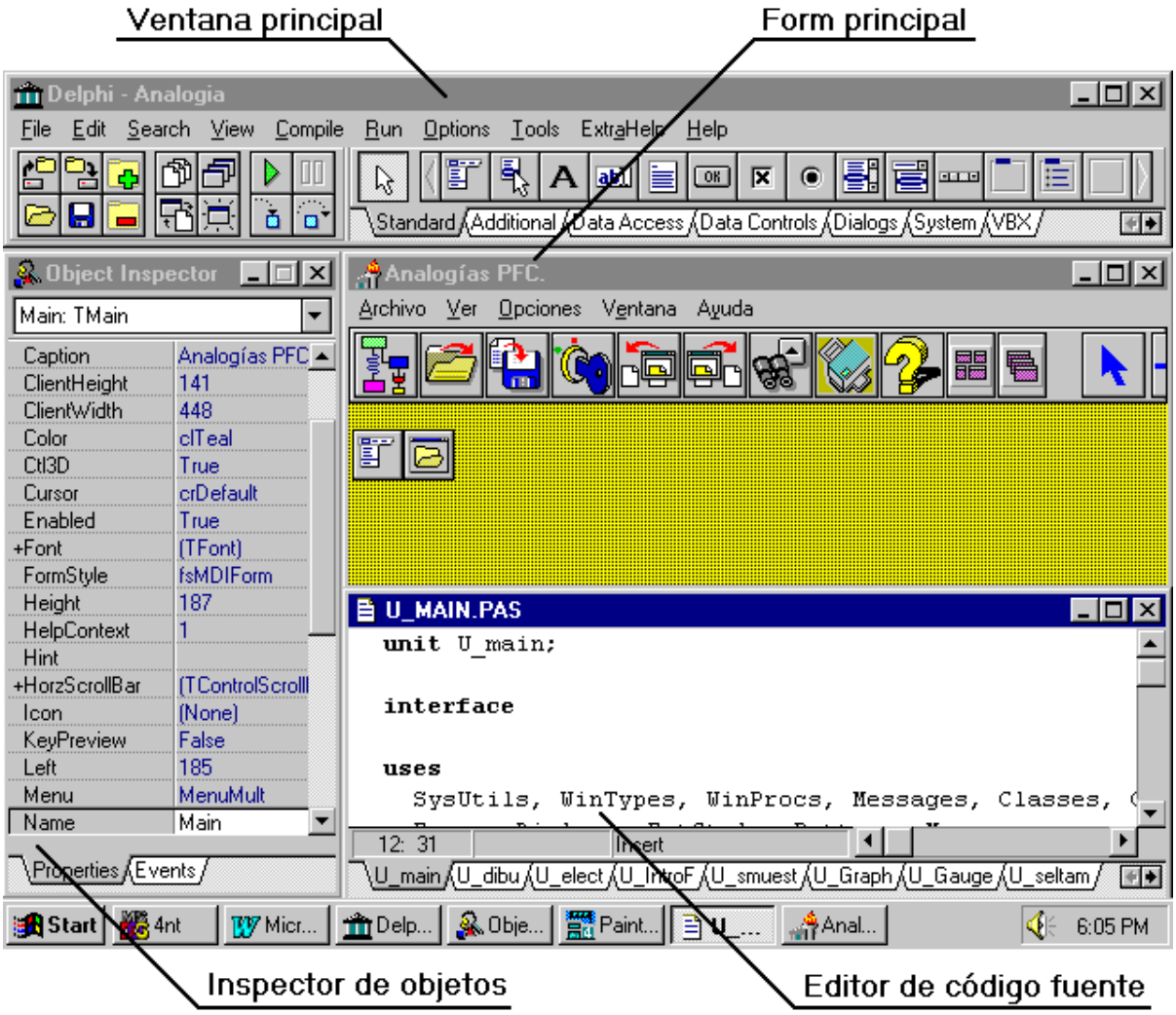


Figura 3.2 Entorno de Delphi con el Form principal de ANALOGIA.EXE

La ventana principal: La barra de programa del margen superior de la pantalla representa la ventana principal de Delphi. Si se cierra, todas las otras ventanas también finalizan su servicio. En la barra de menús de la ventana principal están disponibles todas las órdenes relacionadas con el procesamiento de un proyecto concreto. La carga y almacenamiento de proyectos pertenecen igualmente al menú, así como la presentación u ocultación de las distintas ventanas del entorno de desarrollo.

También se encuentran aquí las órdenes para compilar y ejecutar un programa. Finalmente, desde aquí también se puede llamar a una parte de los programas externos suministrados con el paquete de Delphi: el "Image Editor", el "Database Desktop", y el "BDE Config". El único utilizado en ANALOGIA.EXE es el editor de imágenes, ya que es capaz de crear y editar ficheros .BMP (por ejemplo las imágenes de los componentes mecánicos, etc), .ICO (el icono de la aplicación),

.CUR (Los distintos cursores que he definido), .RES (Resource File) y .DCR (Component Resource).

El mayor espacio de la ventana principal lo ocupa la paleta de componentes, que se encuentra dividida en secciones temáticas a través de unas pestañas. Al igual que en el programa ANALOGIA.EXE, si situamos el cursor sobre cada icono, saldrá un mensaje indicando el tipo de acción que realiza.

Figura3.3 Vista general de la ventana principal

A la izquierda, podemos ver el inspector de objetos, que contiene las "propiedades" y "eventos" del objeto seleccionado en el diseñador. Usted puede hacer click con el mouse en cualquier propiedad para modificarla.

En las propiedades que vienen entre paréntesis, haga doble-click para mostrar un diálogo (por ejemplo, al hacer doble-click a la propiedad (Font), Delphi muestra el diálogo de Tipos de Letra).

Cuando usted extiende la librería con sus propios componentes, usted puede diseñar sus propios editores.

La otra página contiene los eventos, que son los mensajes que Delphi "atrapa" y que usted puede asignar a procedimientos. Cuando diseña sus propios componentes, también puede añadir eventos y relacionarlos con mensajes.

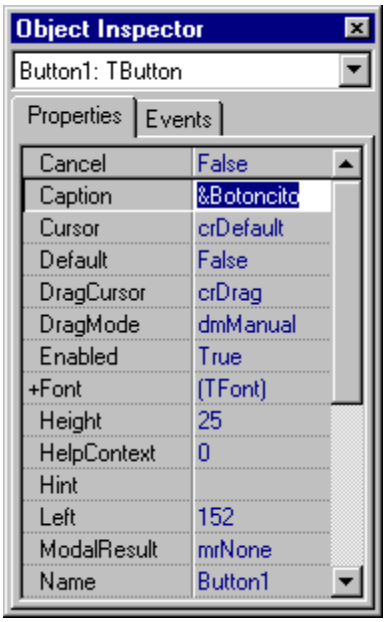


Figura 3.4 Inspector de objetos.

El diseñador nos muestra la(s) forma(s) que podemos diseñar, y puede mostrar tantas formas como usted desee al mismo tiempo. Detrás del diseñador, podemos ver la ventana del editor. El editor es donde escribimos el programa.

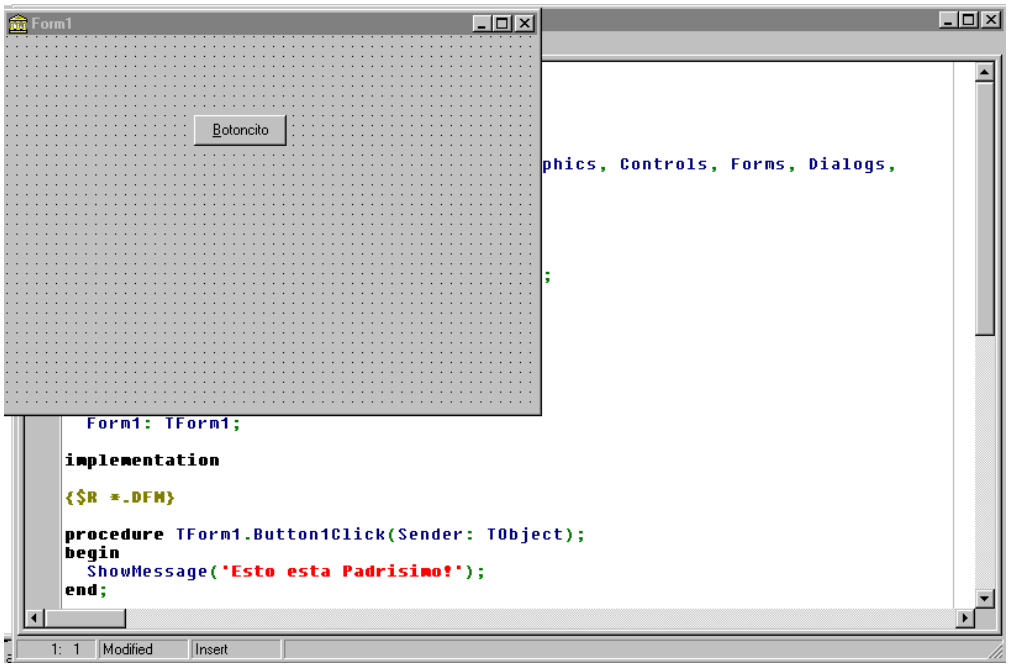


Figura 3.5 Diseñador de objetos.

El editor dispone de una ventana distinta por cada Unit de que disponga el programa, seleccionándose una ventana u otra por medio de pestañas. En concreto, el programa ANALOGIA dispone de once units (de las cuales nueve tienen su form adjunto), cada una de las cuales se selecciona en el editor con su pestaña correspondiente.

Adicionalmente dispone de posibilidad de copia al portapapeles, resaltado de palabras clave, inserción de puntos de ruptura, índices, búsquedas, etc.

EDITOR DE CÓDIGO: Es la parte principal del entorno de desarrollo, es aquí donde se escribirá el código del programa. Por cada formulario que se crea Delphi genera un unit que contiene todo el código asociado a ese formulario. Ese unit se graba con la extensión .Pas. A continuación se muestra el editor de código.

El editor de código fuente: Como se indicó al principio, Delphi es una "Two-Way-Tool", de forma que lo que se va programando visualmente, va apareciendo en forma de código en el editor de código fuente. De la misma forma, si no queremos hacer uso de las capacidades de programación visual, podemos limitarnos a escribir el código (incluida la creación de ventanas, etc.) que se ejecutará.

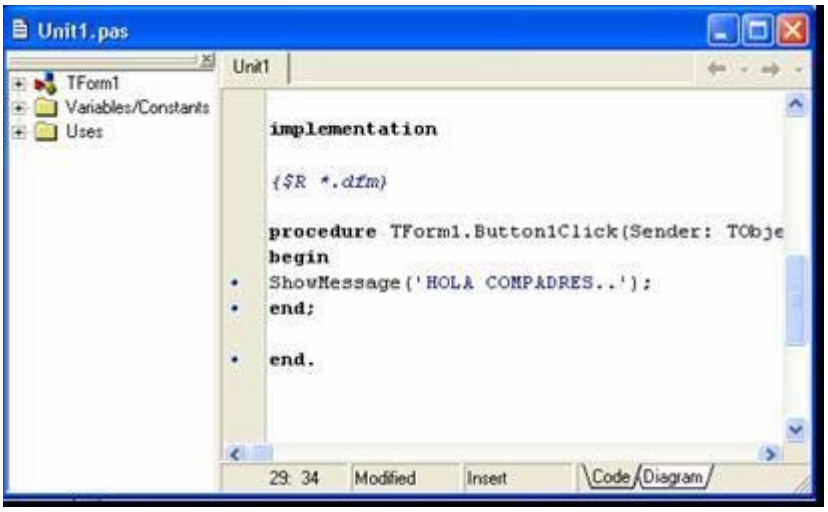


Figura 3.6 Editor de código fuente.

EJEMPLO:

Crear un botón que, cuando se presione, de un mensaje de bienvenida (como en la imagen anterior). Para esto hagamos lo siguiente:

- Seleccione, de la "paleta de componentes", el botón (button).

- Después de seleccionarlo, haga click en donde lo quiera ver en la forma.
- Utilizando la ventana "Object Inspector", cambie la propiedad "Caption", escribiendo "&Botoncito". El caracter "&" le dice a Windows que ponga un subrayado bajo la B y permita al usuario utilizar Alt-B para presionar el Botón.
- Ahora haga doble click en el botón que acaba de crear. Delphi:

Creará un procedimiento en su programa "unit1.pas" llamado Button1Click. En este procedimiento usted especificará lo que va a ocurrir cuando el usuario haga click en el botón. El "begin/end" de pascal especificando donde comienza y termina el botón será también añadido por Delphi.

Asignará el evento "OnClick" del botón (que dice al botón qué procedimiento ejecutar cuando el usuario haga click en el misBmo) al procedimiento "Button1Click" recién creado.

Teclee lo siguiente:

```
ShowMessage('Esto está Padrísimo!');
```

Incluya el punto y coma al final; el lenguaje Pascal necesita saber donde termina cada línea o agrupación de lineas y el símbolo de punto y coma es lo que utiliza para este propósito.

Ahora, presione el botón "Run" de menú de botones de Delphi. Si todo sale bien, Delphi compilará y encadenará su programa. Despues esconderá las ventanas de diseño y desplegará "[Running]" en su barra de título.

Felicidades! Acaba usted de crear, compilar y encadenar un programa de Windows. Veamos lo que puede hacer su programa "de una línea".

3.3.2 Programación orientada a eventos.

Windows se caracteriza por estar dirigida por eventos, de tal forma que un programa no tiene por qué ejecutarse necesariamente de forma secuencial, sino que ciertas porciones de código se ejecutarán cuando ocurra un cierto evento.

Los eventos son señales que el entorno recibe desde distintos elementos, como puedan ser el ratón, el teclado o un temporizador. Estos eventos son redirigidos a las aplicaciones, que en caso de aceptarlos deberán responder adecuadamente de ellos. Ciertos eventos pueden ser gestionados por el propio Windows, otros quedarán a cargo del propio lenguaje que estemos usando, y un tercer grupo serán los que lleguen hasta nuestro programa. En Delphi prácticamente todo el código que escribimos irá asociado a algún evento. Si retomamos el ejemplo del componente para comunicaciones serie, podría interesarnos que se ejecutara un evento cada vez que se recibiese un carácter por el puerto serie, de forma que podríamos escribir el código necesario para guardar el carácter en un archivo cada vez que se produjese el evento. Normalmente los eventos a los que reaccionarán los componentes serán las pulsaciones del teclado o el ratón, activaciones de los componentes, etc.

Los eventos son sucesos que detecta nuestro formulario y según el suceso detectado se ejecuta su código, existen varios eventos, explicaré solo algunos, de cualquier forma durante el curso se explicarán más. Para introducir el código basta con dar doble Click en la parte derecha del evento a utilizar y Delphi automáticamente nos crea el procedimiento de dicho evento y nos muestra el editor de código.

Principales eventos del ratón.

Nombre.	Acción.
OnDlClik	Doble clic de ratón.
OnClick	Un clic de ratón.
OnMouseMove	El ratón pasa por encima del control.
OnMouseDown	Es pulsado un botón del ratón.
OnMouseUp	Es liberado el botón anteriormente pulsado

Tabla 3.1 Principales eventos del ratón.

Principales eventos de teclado.

Nombre	Acción

OnKeyPress	El código que se ponga en este evento se ejecutará cada que se presione una tecla, este evento nos permite revisar la variable Key, misma que contiene contiene la tecla presionada. Previamente la propiedad KeyPreview se tiene que poner en True.
OnKeyDown	Una tecla ha sido pulsada.
OnKeyUp	La tecla anteriormente pulsada ha sido liberada

Tabla 3.2 Principales eventos del teclado.

Principales eventos del sistema.

Nombre	Acción
OnActivate	El formulario se activa, y toma el papel principal.
OnClose	El formulario se cierra.
OnCloseQuery	Se ha solicitado cerrar el formulario.
OnCreate	El formulario se crea.
OnDesactive	El formulario pierde el papel principal pero sigue existiendo.
OnDestroy	El formulario es destruido, y se devuelve la memoria usada a Windows.
OnPaint	El formulario necesita ser pintado, porque algo ha cambiado en él.
OnResize	Ha cambiado el tamaño del formulario.

Tabla 3.3 Principales eventos del sistema.

Nota: Existen eventos que nos ofrecen ciertos datos o variables propios del evento a tratar con los cuales se puede trabajar y pueden ser de mucha utilidad como es el caso de Key del evento OnkeyPress.

3.3.3 Formularios, botones, cajas de texto, etiquetas y menús.

LA VENTANA DE PROGRAMA. FORMULARIOS (FORMS): Delphi hace fácil precisamente las tareas rutinarias, dejando para el programador la tarea realmente importante de codificar el programa, no el entorno.

El entorno (ventanas, colocación de botones, listas, etc) se crea de forma puramente visual, es decir, simplemente se coge con el ratón el componente deseado de la barra de herramientas de la ventana principal y lo sitúa en la ventana sobre la que se desarrolla el programa (Form).

En el Form se puede reproducir el aspecto y el comportamiento de todo tipo de ventanas, simplemente especificando las propiedades correspondientes y/o escribiendo ampliaciones propias. De esta forma se caracteriza a la ventana a través de sus propiedades: anchura, altura, coordenadas, estilo de los bordes, colores, eventos ante los que reacciona, etc.

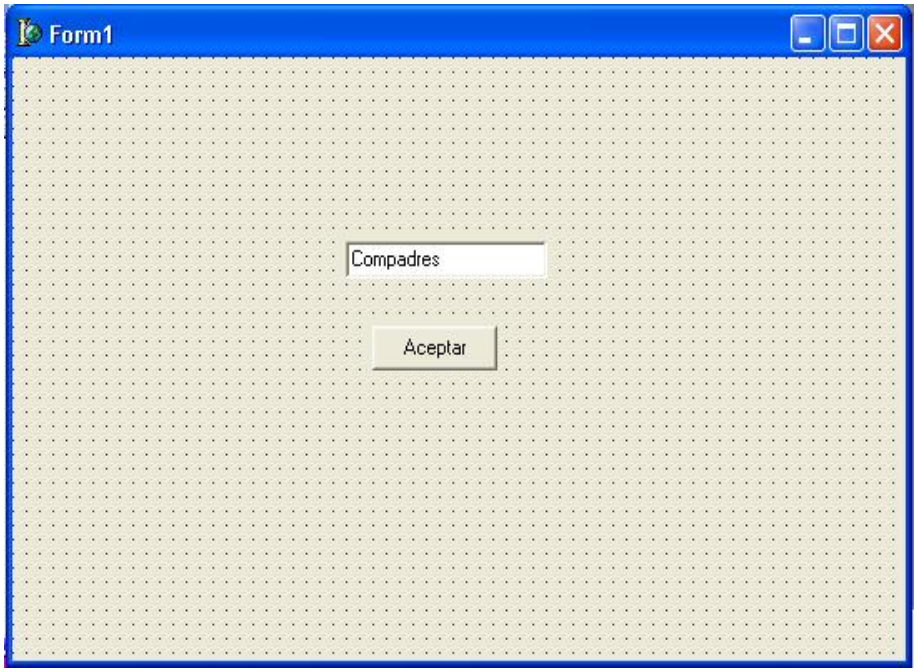


Figura 3.7 La ventana de programa. Formularios (Forms):

EL CONTROL BUTTON: Un botón aparece como una área rectangular que contiene un texto en su interior y que al pulsarlo, lleva a cabo una determinada acción. En Delphi este control aparece en la paleta de componentes con un OK y recibe el nombre de Button. El título que aparece en el interior de un botón corresponde al valor asignado a la propiedad Caption, que al igual que ocurre con las etiquetas de texto, puede contar con un carácter precedido de un &. Este carácter, que aparecerá en el botón subrayando a la letra que le sigue, nos

permite realizar la acción indicada por el botón usando las teclas ALT+LetraSubrayada. En caso de la propiedad Enable del botón tome el valor False, el título aparecerá en un color más difuminado, y el botón no podrá ser pulsado. Tanto el color como el aspecto del título pueden ser controlados mediante la propiedad Font.

EL CONTROL RADIOBUTTON: A veces es necesario dar al usuario a elegir sólo una de las opciones disponibles, creando un grupo de opciones exclusivas entre sí. Estas necesidades serán cubiertas con este control. El aspecto de este control es circular, en lugar de un cuadro, y solo uno de los controles que insertemos en el form podrá estar activado. El título que aparecerá a la derecha del control será facilitado como siempre en la propiedad Caption. El estado actual del botón, seleccionado o no, se conocerá mediante la propiedad Checked, que será Trae en caso afirmativo o False en caso negativo.
ETIQUETAS DE TEXTO O LABEL: Anteriormente utilizamos las etiquetas de texto o Label, lo cual se observo que mediante este control es posible mostrar un texto estático en el form o formulario, fijando su posición, color de letra, tamaño y, obviamente, el texto a mostrar. Tras insertar una etiqueta, lo primero que se debe hacer es asignar un nombre al control, en este caso es necesario modificarlo en el Object Inspector en la propiedad de Name, sin en cambio el valor almacenado en la propiedad Caption será en este caso el texto que se muestra en la etiqueta, también es posible modificar manualmente la posición en la que se encuentra nuestra etiqueta de texto solo seleccionándola y moviéndola de lugar o por otro lado usando valores determinados en las propiedades de Left y Top y las dimensiones con Width (ancho) y Height (alto), se puede ajustar el tamaño y el tipo de letra que se desea y la alineación de la misma con ayuda de la propiedad Size y Font respectivamente. También se puede variar el color de fondo por medio de la propiedad Color el cual aparecerá bajo el texto de la etiqueta.

EL CONTROL EDIT: Uno de los componentes que nos permite la entrada de datos por teclado es el control Edit. El nombre del control lo fijamos en la propiedad Name, al igual que en cualquier otro componente. La posición y dimensiones de un control Edit vienen determinadas por las propiedades Left, Top,Width y Heigth, ya conocidas. Las propiedades Visible y Enable controlan el estado del control, el cual puede ser visible o no y estar activado o no. el color de fondo del campo de entrada será fijado mediante la propiedad Color, mientras que los atributos del texto se establecen mediante la propiedad Font. También es posible fijar una determinada figura para el cursor del ratón al pasar sobre el campo edición, asignando el valor apropiado a la propiedad Cursor. A diferencia del control Label, el control Edit no tiene una propiedad Caption, ya que no dispone de un título estático. El texto que aparece en ese control se puede editar en tiempo de ejecución y se almacena en la propiedad Text. El control Edit recibe eventos generados por el ratón como son: OnMouseMove, OnMouseDown y OnMouseUp, y por el teclado, OnKeyPress, OnKeyDown y OnKeyUp.

EL CONTROL CHECKBOX: Mediante los controles de CheckBox, o cajas de selección. Podemos obtener una entrada de datos. Este control permite al usuario

activar o desactivar una cierta opción sin necesidad de escribir nada, bastará con que realice una pulsación sobre el control. El título que aparecerá junto a la caja de selección será el que asignemos a la propiedad Caption, pudiendo existir una tecla de acceso rápido como en el caso de las etiquetas y los botones. Habitualmente, este control puede aparecer en dos estados distintos, marcado o sin marcar. El estado actual lo podemos conocer mediante la propiedad Checked, que tomará el valor de Trae si está marcado o el valor False en caso contrario.

3.3.4 Elementos adicionales y diálogos.

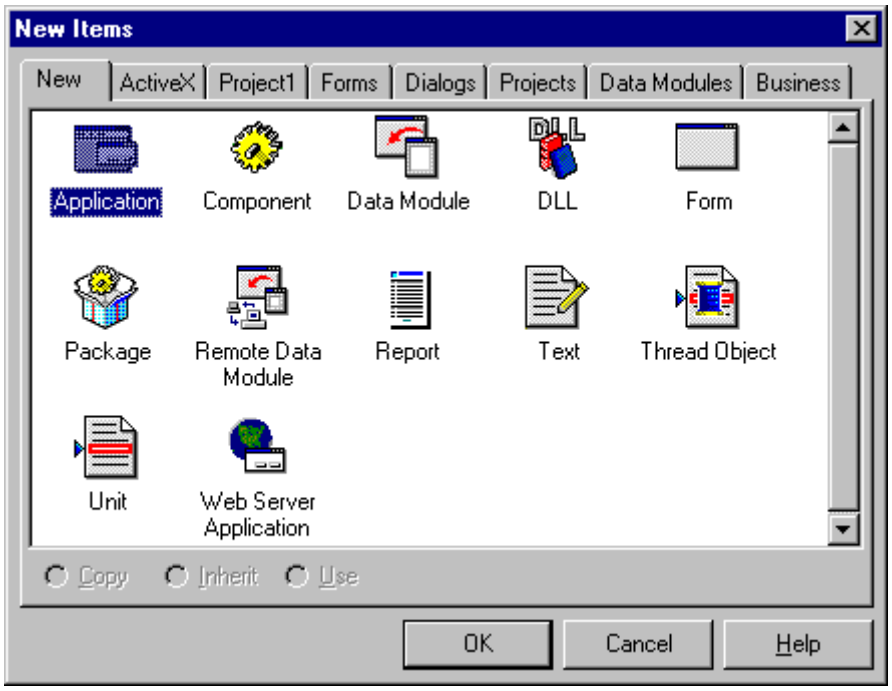
OBJECT REPOSITORY: El Object Repository es un directorio donde usted puede poner código fuente básica y componentes que usted utiliza una y otra vez, como formas, su "aplicación estancar" y cosas así. Cuando usted selecciona File-New, el Code Repository muestra una pagina con todas las opciones de Wizards y lo que contiene el repositorio para que usted elija. Usted puede no solo copiar el código a su proyecto, sino "heredar" y "usar".

Heredar una forma (inherit) es una manera muy util de, por ejemplo, hacer que las formas de su aplicación tengan una apariencia fija. Si usted siempre tiene un Toolbar vacio y su forma siempre tiene el caption rojo, por ejemplo, usted puede heredar de esa forma para todas las formas de su aplicación, y Delphi creara las formas basadas en su forma en vez de basarlas en TForm. Cualquier objeto puede ser heredado. Simplemente "usar" quiere decir que no vamos a mantener una copia del objeto en el repositorio y tampoco lo vamos a utilizar.

WIZARDS Y ADD-INS: Un wizard es un DLL (o DPL) de Delphi que nos ayuda a escribir código. Los Wizards pueden ser desde muy sencillos (wizard para "New Dialog"), hasta muy complejo código que cambia la totalidad del IDE (como Coderush, o GExperts). Cuando son muy complejos y cambian el IDE se llaman "Add-Ins" o IDE Enhancements.

Porque el rango tan amplio de complejidad y son lo mismo? Aunque eso es para otro capitulo, por ahora le puedo decir que Delphi esta hecho en Delphi (y la librería sigue siendo el VCL), así que cuando usted hace un Add-In, todos los objetos del IDE de Delphi están disponibles para que usted los manipule. El IDE de Delphi tiene además interfaces que hacen estas modificaciones faciles, llamadas Open Tools API. Eso quiere decir que su Add-In puede manipular el menu añadiendo items, cajas de dialogo, modificando las ventanas del IDE e incluso reemplazando tareas de Delphi! Este es el poder de un ambiente totalmente abierto a que usted juegue con el. Coderush es el ejemplo mas extremo en el mercado (hasta ahora) de lo que se puede hacer con este API.

UTILIZANDO EL OBJECT REPOSITORY: Comencemos por algo fácil, utilizar uno de los wizards que vienen con Delphi. Para esto no tenemos que saber nada...! Simplemente, seleccione File-New... (No use File-New application porque esto creara una aplicación vacía tal como la que ve cuando comienza Delphi) - la ventana del repositorio de objetos desplegara las opciones de los nuevos elementos que puede crear:



Delphi tiene templates categorizadas por página:

- New - Esta página contiene templates para crear nuevos elementos como Aplicaciones, componentes, modulos de datos, DLLs, Formas, Paquetes de componentes (DPL), Modulos de datos remotos (MIDAS/DCOM/Multi-tier), reportes, threads y servidor de Web.
- ActiveX - Con esta pagina puede usted crear ActiveForms (que funcionan en la pantalla muy parecido a los applets de Java), Control de ActiveX (OCX para usar en Delphi, C++, Visual Basic o VBScript en paginas de web), Automation Object (objeto COM basico sin interface de usuario, pero con conteo de referencia integrado), Pagina de propiedades ActiveX, y Type Library.
- Su Proyecto - El proyecto que se encuentra abierto también aparece como pagina para que usted pueda heredar rápidamente unas formas basadas en otras. Haremos esto mas tarde.
- Forms - Varias formas y reportes diferentes (About, list box, labels, etc).
- Dialogs - Varios dialogos, así como un Wizard que lleva paso a paso por la creación de un dialogo.

- Projects - Wizard para aplicaciones, Aplicación MDI, SDI y aplicación compatible con el logo de Win95 (esta ultima te permite hacer una aplicación básica con los "esqueletos" de todo lo que Microsoft pide para que te ganes un logo de "Diseñado para Windows 95".
- Data Modules- Los modulos de datos que haga usted pueden ir aqui. En la seccion de bases de datos veremos como funcionan los modulos de datos.
- Business - Esta pagina tiene un wizard para hacer una forma con campos (lista para ejecutar) a partir de una base de datos, un "cubo de decisiones", un wizard para hacer reportes, y un wizard para graficación.

Cuando use Delphi para producir sus aplicaciones, podrá usar Tools-Repository del menu de Delphi para hacer mas paginas y organizar sus wizards o los modulos de datos que haga. Además, usando Tools-Environment options - Shared repository directory podrá usted cambiar el directorio para que este en su directorio de datos, o en algún lugar de la red (asegurese de copiar los archivos de C:\Program Files\Borland\Delphi 3\ObjRepos, o equivalente, al nuevo directorio para que los wizards sigan funcionando).

3.3.5 Compilación y depuración de programas.

Un proyecto Delphi es el conjunto de todos los archivos que, una vez compilados, constituyen un programa ejecutable.

En el caso de ANALOGIA.EXE solo es necesario este archivo (exceptuando ANALOGIA.INI y ANALOGIA.HLP que ofrecen prestaciones adicionales), pero en otros casos es necesario añadir otros ficheros: DLLs, VBXs, BDE, etc.

Los fuentes de ANALOGIA.EXE se componen de un archivo ANALOGIA.DPR (DPR = Delphi Project) que es el programa principal (no la unit Main, sino el archivo que únicamente crea todos los forms y los lanza, siendo el archivo más pequeño de todos) y una serie de units (once en nuestro caso) que suelen llevar extensión .PAS. Generalmente esta unidades se encuentran asociados con fichas, aunque no tiene por qué ser así en todos los casos (en ANALOGIA.EXE las unidades U_Misc y U_Math no están asociadas a ningún form).

Cuando se compila el proyecto, se crean una serie de módulos de código objeto (.DCU), que el linker utiliza para crear el archivo .EXE.

Otro tipo de archivos muy importante en la programación bajo Windows son los archivos de recursos (.RES), generados tanto por Delphi como por otros compiladores y herramientas (C++, etc). Estos ficheros pueden contener mapas de bits, listas de strings, etc. De hecho, un form es también del tipo Ressource

(pero con extensión .DFM, o sea Delphi Form). Para la vinculación en un módulo del programa se utiliza la instrucción de compilación {$R <NombreArchivo.RES>}, que podemos ver al Bprincipio del programa. De hecho, los dibujos de los cursores, los dibujos de los componentes eléctricos y mecánicos y otra serie de elementos los he creado en archivos .RES independientes e importado posteriormente al programa.

Aunque no son muy utilizados, también se pueden incluir archivos .INC (Include) que contienen clases, rutinas, variables, etc. y archivos .OBJ (Objeto), generados por ejemplo por un ensamblador o un compilador de C.

3.4 Aplicaciones Generales

Es una potente herramienta de desarrollo de programas que permite la creación de aplicaciones para Windows 3.x, Windows 95 y Windows NT. Dispone de un compilador muy rápido (más que la mayoría de los compiladotes de C++, como ya era tradicional en Turbo Pascal), y potentes herramientas para la creación visual de aplicaciones de completas herramientas para la creación y manejo de bases de datos, aplicaciones multimedia, enlace DDE, creación de DLLs, VBX, etc. Cubre muchos temas de programación bajo Windows: se incluye entre los mismos un completo centro de control para la creación de aplicaciones multimedia, así como una gran variedad de componentes que actúan “debajo” del entorno, como tipos de listado muy variados y contenedores generales de datos. Las aplicaciones terminadas están disponibles en archivos ejecutables (EXE) que peden utilizarse sólo con bibliotecas adicionales.

3.5 Conclusiones

No cabe duda que ante la cuota de mercado y los premios obtenidos, Delphi tiene un futuro prometedor, además Inprise intenta que cada nueva versión incorpore los nuevos estándares del mercado con los que nos aseguramos siempre una puesta al día. Y una cosa muy importante, que gracias a los componentes y tecnologías incorporados, tenemos todo lo suficiente para el desarrollo de programas de gestión y web, sin recurrir a librerías de terceros.

CAPÍTULO 4

4. PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

4.1 Objetivo

Proporcionar los elementos para entender y utilizar la Programación Orientada a Objetos (POO), a través del lenguaje de programación JAVA.

4.2 Introducción

El lenguaje de programación Java fue desarrollado por SUN con la intención de competir con Microsoft en el mercado de la red.

El éxito de Java reside en varias de sus características. Java es un lenguaje sencillo, o todo lo sencillo que puede ser un lenguaje orientado a objetos. Es un lenguaje independiente de plataforma, por lo que un programa hecho en Java se ejecutará igual en un PC con Windows que en una estación de trabajo basada en Unix. También hay que destacar su seguridad, desarrollar programas que accedan ilegalmente a la memoria o realizar caballos de Troya es una tarea propia de titanes.

Cabe mencionar también su capacidad multihilo, su robustez o lo integrado que tiene el protocolo TCP/IP, lo que lo hace un lenguaje ideal para Internet. Pero es su sencillez, portabilidad y seguridad lo que le han hecho un lenguaje de tanta importancia.

Java además cuenta con la portabilidad la cual hace de Java un lenguaje medio interpretado y medio compilado. Pues se toma el código fuente, se compila a un lenguaje intermedio cercano al lenguaje máquina pero independiente del ordenador y el sistema operativo en que se ejecuta (conocido como bytecodes) y, finalmente, se interpreta ese lenguaje intermedio por medio de un programa denominado máquina virtual de Java.

4.3 Desarrollo

Durante el desarrollo de este capitulo veremos algunas de las principales caracteristicas del lenguaje de Programación Java.

4.3.1 Conceptos de Programación Orientada a Objetos.

Características de Java.

Las características principales que nos ofrece Java respecto a cualquier otro lenguaje de programación, son:

- Simple. Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++.
- Orientado a Objetos. Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo
- Distribuido. Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.
- Robusto. Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error.
- Arquitectura Neutral. Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado.
- Seguro. El código Java pasa muchos tests antes de ejecutarse en una máquina. El código se pasa a través de un verificador de byte-codes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal -código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto. las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus.

- Interpretado. Java para conseguir ser un lenguaje independiente del sistema operativo y del procesador que incorpore la máquina utilizada, es tanto interpretado como compilado. Con este sistema es fácil crear aplicaciones multiplataforma, pero para ejecutarlas es necesario que exista el run-time correspondiente al sistema operativo utilizado.
- Dinámico. Java se beneficia todo lo posible de la tecnología orientada a objetos. Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución. Las libreria nuevas o actualizadas no paralizarán las aplicaciones actuales (siempre que mantengan el API anterior). La figura 4.1 muestra como trabaja el navegador de Java.

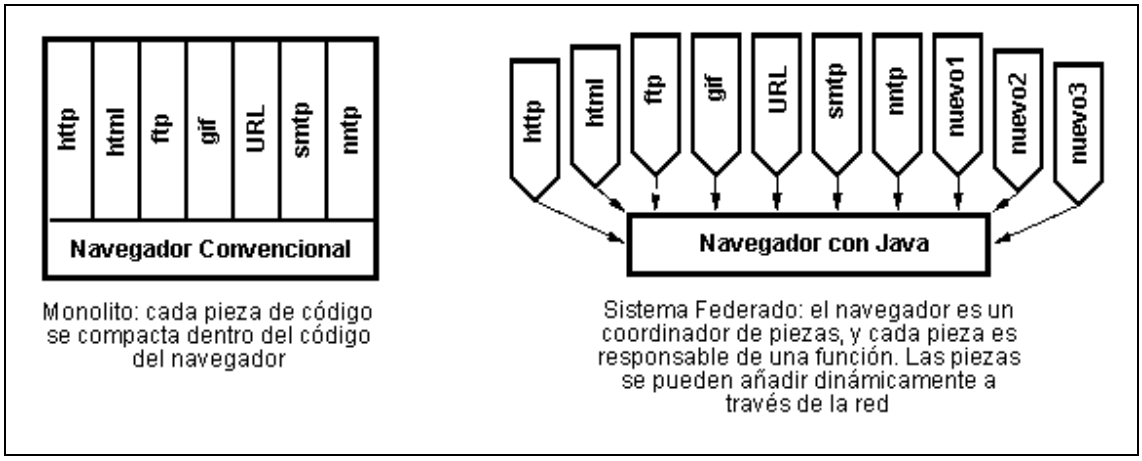


Figura 4.1. Navegador de Java.

El lenguaje de programación Java esta basado principalmente en la programación orientada a objetos, por lo tanto antes de comenzar de lleno con lo que es Java en su totalidad, veremos un poco a cerca de los conceptos de la programación orientada a objetos.

PROGRAMACIÓN ORIENTADA A OBJETOS.

Dado que Java es un lenguaje orientado a objetos, es imprescindible entender qué es esto y en qué afecta a nuestros programas. Desde el principio, la carrera por crear lenguajes de programación ha sido una carrera para intentar realizar abstracciones sobre la máquina. Al principio no eran grandes abstracciones y el concepto de lenguajes imperativos es prueba de ello. Exigen pensar en términos del ordenador y no en términos del problema a solucionar. Esto provoca que los programas sean difíciles de crear y mantener, al no tener una relación obvia con el problema que representan No abstraen lo suficiente.

Muchos paradigmas de programación intentaron resolver este problema alterando la visión del mundo y adaptándola al lenguaje. Estas aproximaciones modelaban el mundo como un conjunto de objetos o de listas. Funcionaban bien para algunos problemas pero no para otros. Los lenguajes orientados a objetos, más generales, permiten realizar soluciones que, leídas, describen el problema. Permiten escribir soluciones pensando en el problema y no en el ordenador que debe solucionarlo en último extremo. Se basan en cinco características:

- Todo es un objeto. Cada elemento del problema debe ser modelizado como un objeto.
- Un programa es un conjunto de objetos diciéndose entre sí que deben hacer por medio de mensajes. Cuando necesitas que un objeto haga algo le mandas un mensajes. Más concretamente, ejecutas un método de dicho objeto.
- Cada objeto tiene su propia memoria, que llena con otros objetos. Cada objeto puede contener otros objetos. De este modo se puede incrementar la complejidad del programa, pero detrás de dicha complejidad sigue habiendo simples objetos.
- Todo objeto tiene un tipo. En POO, cada objeto es una instancia (un caso particular) de una clase (el tipo general). Lo que distingue a una clase de otra es la respuesta a la pregunta, ¿qué mensajes puedes recibir?

Todos los objetos de un determinado tipo pueden recibir los mismos mensajes. Por ejemplo, dado que un objeto de tipo Gato es también un objeto de tipo Animal, se puede hacer código pensando en los mensajes que se mandan a un animal y aplicarlo a todos los objetos de ese tipo, sin pensar si son también gatos o no.

CREAR OBJETOS EN JAVA

En Java, se crea un objeto mediante la creación de un objeto de una clase o, en otras palabras, ejemplarizando una clase.

Hasta entonces, los ejemplos que se muestran aquí crean objetos a partir de clases que ya existen en el entorno Java.

Frecuentemente, se verá la creación de un objeto Java con una sentencia como esta.

```
Date hoy = new Date();
```

Esta sentencia crea un objeto Date (Date es una clase del paquete java,util). Esta sentencia realmente realiza tres acciones: declaración, ejemplarización e inicialización.

Date hoy es una declaración de variable que sólo le dice al compilador que el nombre hoy se va a utilizar para referirse a un objeto cuyo tipo es Date, el operador new ejemplariza la clase Date (creando un nuevo objeto Date), y Date() inicializa el objeto.

4.3.2 Clases y objetos.

¿Qué son los objetos?

En informática, un OBJETO es un conjunto de variables y de los métodos relacionados con esas variables.

Un poco más sencillo: un objeto contiene en sí mismo la información y los métodos o funciones necesarios para manipular esa información.

Lo más importante de los objetos es que permiten tener un control total sobre 'quién' o 'qué' puede acceder a sus miembros, es decir, los objetos pueden tener miembros públicos a los que podrán acceder otros objetos o miembros privados a los que sólo puede acceder él. Estos miembros pueden ser tanto variables como funciones.

El gran beneficio de todo esto es la encapsulación, el código fuente de un objeto puede escribirse y mantenerse de forma independiente a los otros objetos contenidos en la aplicación.

¿Qué son las clases?

Una CLASE es un proyecto, o prototipo, que define las variables y los métodos comunes a un cierto tipo de objetos.

Un poco más sencillo: las clases son las matrices de las que luego se pueden crear múltiples objetos del mismo tipo. La clase define las variables y los métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Primero deberemos crear una clase antes de poder crear objetos o ejemplares de esa clase.

¿Qué son los mensajes?

Para poder crear una aplicación se necesita más de un objeto, y estos objetos no pueden estar aislados unos de otros, pues bien, para comunicarse esos objetos se envían mensajes.

Los mensajes son simples llamadas a las funciones o métodos del objeto con el se quiere comunicar para decirle que haga cualquier cosa.

HERENCIA

Qué significa la herencia, quién hereda qué; esto sólo significa que se puede crear una clase partiendo de otra que ya exista.
Es decir, se puede crear una clase a través de una clase existente, y esta clase tendrá todas las variables y los métodos de su 'superclase', y además se le podrán añadir otras variables y métodos propios. Se llama 'Superclase' a la clase de la que desciende una clase.

DECLARAR UN OBJETO

Ya que la declaración de un objeto es una parte innecesaria de la creación de un objeto, las declaraciones aparecen frecuentemente en la misma línea que la creación del objeto. Como cualquier otra declaración de variable, las declaraciones de objetos pueden aparecer solitarias como esta.

Date hoy;

De la misma forma, declarar una variable para contener un objeto es exactamente igual que declarar una variable que va a contener un tipo primitivo.

tipo nombre

donde tipo es el tipo de dato del objeto y nombre es el nombre que va a utilizar el objeto. En Java, las clases e interfaces son como tipos de datos. Entonces tipo puede ser el nombre de una clase o de un interface.

Las declaraciones notifican al compilador que se va a utilizar nombre para referirse a una variable cuyo tipo es tipo. Las declaraciones no crean nuevos objetos. Date hoy no crea un objeto Date, sólo crea un nombre de variable para contener un objeto Date. Para ejemplarizar la clase Date, o cualquier otra clase, se utiliza el operador new.

EJEMPLARIZAR UNA CLASE

El operador new ejemplariza una clase mediante la asignación de memoria para el objeto nuevo de ese tipo. new necesita un sólo argumento: una llamada al método constructor. Los métodos constructores son métodos especiales proporcionados por cada clase Java que son responsables de la inicialización de los nuevos objetos de ese tipo. El operador new crea el objeto, el constructor lo inicializa. Aquí tienes un ejemplo del uso del operador new para crear un objeto Rectangle.

new Rectangle(0, 0, 100, 200);

En el ejemplo, Rectangle(0, 0, 100, 200) es una llamada al constructor de la clase Rectangle.

El operador new devuelve una referencia al objeto recién creado. Esta referencia puede ser asignada a una variable del tipo apropiado.

Rectangle rect = new Rectangle(0, 0, 100, 200);

Recuerda que una clase esencialmente define un tipo de dato de referencia. Por eso, Rectangle puede utilizarse como un tipo de dato en los programas Java. El valor de cualquier variable cuyo tipo sea un tipo de referencia, es una referencia (un puntero) al valor real o conjunto de valores representado por la variable.
INICIALIZAR UN OBJETO

Como se mocionó anteriormente, las clases proporcionan métodos constructores para inicializar los nuevos objetos de ese tipo. Una clase podría proporcionar múltiples constructores para realizar diferentes tipos de inicialización en los nuevos objetos.

Cuando se vea la implementación de una clase, se reconocerán los constructores porque tienen el mismo nombre que la clase y no tienen tipo de retorno. Recuerda la creación del objeto Date en el sección inicial. El constructor utilizado no tenía ningún argumento.

Date()

Un constructor que no tiene ningún argumento, como el mostrado arriba, es conocido como constructor por defecto. Al igual que Date, la mayoría de las clases tienen al menos un constructor, el constructor por defecto.

Si una clase tiene varios constructores, todos ellos tienen el mismo nombre pero se deben diferenciar en el número o el tipo de sus argumentos. Cada constructor inicializa el nuevo objeto de una forma diferente. Junto al constructor por defecto, la clase Date proporciona otro constructor que inicializa el nuevo objeto con un nuevo año, mes y día.

Date cumpleaños = new Date(1963, 8, 30);

El compilador puede diferenciar los constructores a través del tipo y del número de sus argumentos.

REFERENCIAR VARIABLES DE UN OBJETO

Primero, se enfoca en cómo inspeccionar y modificar la posición del rectángulo mediante la manipulación directa de las variables x e y. La siguiente sección mostrará como mover el rectángulo llamando al método move().

Para acceder a las variables de un objeto, sólo se tiene que añadir el nombre de la variable al del objeto referenciado introduciendo un punto en el medio ('.').

objetoReferenciado.variable

Suponiendo que se tiene un rectángulo llamado rect en tu programa. puedes acceder a las variables x e y con rect.x y rect.y, respectivamente.

Ahora que ya tienes un nombre para las variables de rect, puedes utilizar ese nombre en sentencias y expresiones Java como si fueran nombres de variables "normales". Así, para mover el rectángulo a una nueva posición podrías escribir.

```
rect.x = 15;      // cambia la posición x
rect.y = 37;      // cambia la posición y
```

La clase Rectangle tiene otras dos variables width y height que son accesibles para objetos fuera de Rectangle. Se puede utilizar la misma notación con ellas: rect.width y rect.height. Entonces se puede calcular el área del rectángulo utilizando esta sentencia.

```
area = rect.height * rect.width;
```

Cuando se accede a una variable a través de un objeto, se está refiriendo a las variables de un objeto particular. Si cubo fuera también un rectángulo con una altura y anchura diferentes de rect, esta instrucción.

```
area = cubo.height * cubo.width;
```

calcula el área de un rectángulo llamado cubo y dará un resultado diferente que la instrucción anterior (que calculaba el área de un rectángulo llamado rect).

Observa que la primera parte del nombre de una variable de un objeto (el objetoReferenciado en objetoReferenciado.variable debe ser una referencia a un objeto. Como se puede utilizar un nombre de variable aquí, también se puede utilizar en cualquier expresión que devuelva una referencia a un objeto. Recuerda que el operador new devuelve una referencia a un objeto. Por eso, se puede utilizar el valor devuelto por new para acceder a las variables del nuevo objeto.

```
height = new Rectangle().height;
```

LLAMAR A MÉTODOS DE UN OBJETO

Llamar a un método de un objeto es similar a obtener una variable del objeto. Para llamar a un método del objeto, simplemente se añade al nombre del objeto referenciado el nombre del método, separados por un punto ('.'), y se proporcionan los argumentos del método entre paréntesis. Si el método no necesita argumentos, se utilizan los paréntesis vacíos.

```
objetoReferenciado.nombreMétodo(listaArgumentos);
  o
objetoReferenciado.nombreMétodo();
```

Que significa esto en términos de movimiento del rectángulo. Para mover rect a una nueva posición utilizando el método move() se escribe esto.

```
rect.move(15, 37);
```

Esta sentencia Java llama al método move() de rect con dos parámetros enteros, 15 y 37. Esta sentencia tiene el efecto de mover el objeto rect igual que se hizo en las sentencias anteriores en las que se moficaban directamente los valores x e y del objeto.

```
rect.x = 15;
rect.y = 37;
```

Si se quiere mover un rectángulo diferente, uno llamado cubo, la nueva posición se podría escribir.

```
cubo.move(244, 47);
```

Como se ha visto en estos ejemplos, las llamadas a métodos se hacen directamente a un objeto específico; el objeto especificado en la llamada al método es el que responde a la instrucción.

Las llamadas a métodos también se conocen como mensajes. Como en la vida real, los mensajes se deben dirigir a un receptor particular. Se pueden obtener distintos resultados dependiendo del receptor de su mensaje.

En el ejemplo anterior, se ha enviado el mensaje move() al objeto llamado rect para que éste mueva su posición.
Cuando se envía el mensaje move() al objeto llamado cubo, el que se mueve es cubo. Son resultados muy distintos.

Una llamada a un método es una expresión y evalúa a algún valor. El valor de una llamada a un método es su valor de retorno, si tiene alguno. Normalmente se asignará el valor de retorno de un método a una variable o se utilizará la llamada al método dentro del ámbito de otra expresión o sentencia.

El método move() no devuelve ningún valor (está declarado como void). Sin embargo, el método inside() de Rectangle si lo hace. Este método toma dos coordendas x e y, y devuelte true si este punto está dentro del rectángulo.

Se puede utilizar el método inside() para hacer algo especial en algún punto, como decir la posición del ratón cuando está dentro del rectángulo.

```
if (rect.inside(mouse.x, mouse.y)) {
      // ratón dentro del rectángulo
} else {
      // ratón fuera del rectángulo
}
```

Recuerda que una llamada a un método es un mensaje al objeto nombrado. En este caso, el objeto nombrado es rect. Entonces.

```
rect.inside(mouse.x, mouse.y)
```

le pregunta a rect si la posición del cursor del ratón se encuentra entre las coordenadas mouse.x y mouse.y. Se podría obtener una respuesta diferente si envía el mismo mensaje a cubo.

Como se explicó anteriormente, el objetoReferenciado en la llamada al método objetoReferenciado.metodo() debe ser una referencia a un objeto. Como se puede utilizar un nombre de variable aquí, también se puede utilizar en cualquier expresión que devuelva una referencia a un objeto. Recuerda que el operador new devuelve una referencia a un objeto. Por eso, se puede utilizar el valor devuelto por new para acceder a las variables del nuevo objeto.

```
new Rectangle(0, 0, 100, 50).equals(anotherRect)
```

La expresión new Rectangle(0, 0, 100, 50) evalúa a una referencia a un objeto que se refiere a un objeto Rectangle.

Entonces, como se observa, se puede utilizar la notación de punto ('.') para llamar al método equals() del nuevo objeto Rectangle para determinar si el rectángulo nuevo es igual al especificado en la lista de argumentos de equals().

DECLARAR CLASES EN JAVA

Ahora que ya se sabe como crear, utilizar objetos, es hora de aprender cómo escribir clases de las que crear esos objetos.

Una clase es un proyecto o prototipo que se puede utilizar para crear muchos objetos. La implementación de una clase comprende dos componentes: la declaración y el cuerpo de la clase.

```
DeclaraciónDeLaClase {
  CuerpoDeLaClase
  }
```

Como mínimo, la declaración de una clase debe contener la palabra clave class y el nombre de la clase que está definiendo. Así la declaración más sencilla de una clase se parecería a esto.

```
class NombredeClase {
}
```

Por ejemplo, esta clase declara una nueva clase llamada NumeroImaginario.

```
class NumeroImaginario {
}
```

Los nombres de las clases deben ser un identificador legal de Java y, por convención, deben empezar por una letra mayúscula. Muchas veces, todo lo que se necesitará será una declaración mínima. Sin embargo, la declaración de una clase puede decir más cosas sobre la clase. Más específicamente, dentro de la declaración de la clase se puede.

- declarar cual es la superclase de la clase.
- listar los interfaces implementados por la clase
- declarar si la clase es pública, abstracta o final

DECLARAR LA SUPERCLASE DE LA CLASE

En Java, todas las clases tienen una superclase. Si no se especifica una superclase para una clase, se asume que es la clase Object (declarada en java.lang). Entonces la superclase de NumeroImaginario es Object porque la declaración no explicitó ninguna otra clase.

Para especificar explícitamente la superclase de una clase, se debe poner la palabra clave extends más el nombre de la superclase entre el nombre de la clase que se ha creado y la llave abierta que abre el cuerpo de la clase, así.

```
class NombredeClase extends NombredeSuperClase {
  }
```

Por ejemplo, si se quiere que la superclase de NumeroImaginario sea la clase Number en vez de la clase Object. Se podría escribir esto.

```
class NumeroImaginario extends Number {
}
```

Esto declara explícitamente que la clase Number es la superclase de NumeroImaginario. (La clase Number es parte del paquete java.lang y es la base para los enteros, los números en coma flotante y otros números).

Declarar que Number es la superclase de NumeroImaginario declara implícitamente que NumeroImaginario es una subclase de Number. Una subclase hereda las variables y los métodos de su superclase.

4.3.3 Tipos de datos. Operadores y expresiones.

VARIABLES Y TIPOS DE DATOS

Las variables son las partes importantes de un lenguaje de programación: estas son las entidades (valores, datos) que actúan y sobre las que se actúa.

Una declaración de variable siempre contiene dos componentes, el tipo de la variable y su nombre.

```
tipoVariable nombre;
```

Todas las variables en el lenguaje Java deben tener un tipo de dato. El tipo de la variable determina los valores que la variable puede contener y las operaciones que se pueden realizar con ella.

Existen dos categorías de datos principales en el lenguaje Java: los tipos primitivos y los tipos referenciados.

Los tipos primitivos contienen un sólo valor e incluyen los tipos como los enteros, coma flotante, los caracteres, etc... En la Tabla 4.1 están todos los tipos primitivos soportados por el lenguaje Java, su formato, su tamaño y una breve descripción de cada uno.

Tipo	Tamaño/Formato	Descripción
(Números enteros)		
byte	8-bit complemento a 2	Entero de un Byte
short	16-bit complemento a 2	Entero corto
int	32-bit complemento a 2	Entero
long	64-bit complemento a 2	Entero largo
(Números reales)		
float	32-bit IEEE 754	Coma flotante de precisión simple
double	64-bit IEEE 754	Coma flotante de precisión doble
(otros tipos)		
char	16-bit Caracter	Un sólo carácter
boolean	true o false	Un valor booleano (verdadero o falso)

Tabla 4.1. Variables y Tipos de Datos.

Los tipos referenciados se llaman así porque el valor de una variable de referencia, es una referencia (un puntero) hacia el valor real. En Java tenemos los arrays, las clases y los interfaces como tipos de datos referenciados.

NOMBRES DE VARIABLES

Un programa se refiere al valor de una variable por su nombre. Por convención, en Java, los nombres de las variables empiezan con una letra minúscula (los nombres de las clases empiezan con una letra mayúscula).

Un nombre de variable Java.

1. Debe ser un identificador legal de Java comprendido en una serie de caracteres Unicode. Unicode es un sistema de codificación que soporta texto escrito en distintos lenguajes humanos. Unicode permite la codificación de 34.168 caracteres. Esto le permite utilizar en sus

programas Java varios alfabetos como el Japonés, el Griego, el Ruso o el Hebreo. Esto es importante para que los programadores pueden escribir código en su lenguaje nativo.

2. No puede ser el mismo que una palabra clave o el nombre de un valor booleano (true or false)
3. No deben tener el mismo nombre que otras variables cuyas declaraciones aparezcan en el mismo ámbito.

La regla número 3 implica que podría existir el mismo nombre en otra variable que aparezca en un ámbito diferente.

Por convención, los nombres de variables empiezan por una letra minúscula. Si una variable está compuesta de más de una palabra, como 'nombreDato' las palabras se ponen juntas y cada palabra después de la primera empieza con una letra mayúscula.

OPERADORES

Los operadores realizan algunas funciones en uno o dos operandos. Los operadores que requieren un operador se llaman operadores unarios. Por ejemplo, ++ es un operador unario que incrementa el valor su operando en uno.
Los operadores que requieren dos operandos se llaman operadores binarios. El operador = es un operador binario que asigna un valor del operando derecho al operando izquierdo.

Los operadores unarios en Java pueden utilizar la notación de prefijo o de sufijo. La notación de prefijo significa que el operador aparece antes de su operando.

operador operando

La notación de sufijo significa que el operador aparece después de su operando:

operando operador

Además de realizar una operación también devuelve un valor. El valor y su tipo dependen del tipo del operador y del tipo de sus operandos. Por ejemplo, los operadores aritméticos (realizan las operaciones de aritmética básica como la suma o la resta) devuelven números, el resultado típico de las operaciones aritméticas. El tipo de datos devuelto por los operadores aritméticos depende del tipo de sus operandos: si sumas dos enteros, obtendrás un entero. Se dice que una operación evalúa su resultado.

Es muy útil dividir los operadores Java en las siguientes categorías: aritméticos, relacionales y condicionales. Lógicos y de desplazamiento, y de asignación.

OPERADORES ARITMÉTICOS

El lenguaje Java soporta varios operadores aritméticos - incluyendo + (suma), - (resta), * (multiplicación), / (división), y % (módulo), en todos los números enteros y de coma flotante. Por ejemplo, puedes utilizar este código Java para sumar dos números:

sumaEsto + aEsto
O este código para calcular el resto de una división:
divideEsto % porEsto
En la Tabla 4.2 muestra todas las operaciones aritméticas binarias en Java.

Operador	Uso	Descripción
+	op1 + op2	Suma op1 y op2
-	op1 - op2	Resta op2 de op1
*	op1 * op2	Multiplica op1 y op2
/	op1 / op2	Divide op1 por op2
%	op1 % op2	Obtiene el resto de dividir op1 por op2

Tabla 4.2. Operaciones Aritméticas.

Nota: El lenguaje Java extiende la definición del operador + para incluir la concatenación de cadenas.

Los operadores + y - tienen versiones unarias que seleccionan el signo del operando. (Tabla 4.3)

Operador	Uso	Descripción
+	+ op	Indica un valor positivo
-	- op	Niega el operando

Tabla 4.3. Operadores Unarios.

Además, existen dos operadores de atajos aritméticos, ++ que incrementa en uno su operando, y -- que decrementan en uno el valor de su operando. (Tabla 4.4).

Operador	Uso	Descripción
++	op ++	Incrementa op en 1; evalúa el valor antes de incrementar
++	++ op	Incrementa op en 1; evalúa el valor después de incrementar
--	op --	Decrementa op en 1; evalúa el valor antes de decrementar
--	-- op	Decrementa op en 1; evalúa el valor después de decrementar

Tabla 4.4. Operadores de Incremento y Decremento.

4.3.4 Sentencias de control de flujo del programa.

Sentencias de Control de Flujo en Java.

Las sentencias de control de flujo determinan el orden en que se ejecutarán las otras sentencias dentro del programa. El lenguaje Java soporta varias sentencias de control de flujo, incluyendo. (Tabla 4.5)

Sentencias	palabras clave
toma de decisiones	if-else, switch-case
bucles	for, while, do-while
excepciones	try-catch-finally, throw

Tabla 4.5. Sentencias de Control de Flujo en Java.

SENTENCIA IF-ELSE.

La sentencia if-else de java proporciona a los programas la posibilidad de ejecutar selectivamente otras sentencias basándose en algún criterio.

Por ejemplo, un programa imprime información de depurado basándose en el valor de una variable booleana llamada DEBUG. Si DEBUG fuera verdadera true, el programa imprimiría la información de depurado, como por ejemplo, el valor de una variable como X. Si DEBUG es false el programa procederá normalmente. Un segmento de código que implemente esto se podría parecer a este.

```
. . .

if (DEBUG)
    System.out.println("DEBUG: x = " + x);
. . .
```

Esta es la versión más sencilla de la sentencia if: la sentencia gobernada por if se ejecuta si alguna condición es verdadera. Generalmente, la forma sencilla de if se puede escribir así.

```
if (expresión)
    sentencia
```

Existe otra forma de la sentencia else, else if que ejecuta una sentencia basada en otra expresión. Por ejemplo, se requiere de un programa que asigna notas basadas en la puntuación de un examen, un sobresaliente para una puntuación del 90% o superior, un notable para el 80% o superior y demás. Se podría utilizar una sentencia if con una serie de comparaciones else if y una sentencia else para escribir este código.

```
int puntuación;
String nota;
if (puntuación >= 90) {
    nota = "Sobresaliente";
} else if (puntuacion >= 80) {
    nota = "Notable";
} else if (puntuacion >= 70) {
    nota = "Bien";
} else if (puntuacion >= 60) {
    nota = "Suficiente";
} else {
    nota = "Insuficiente";
}
```

Una sentencia if puede tener cualquier número de sentencias de acompañamiento else if.

Se puede observar que algunos valores de puntuación pueden satisfacer más una de las expresiones que componen la sentencia if. Por ejemplo, una puntuación de 76 podría evaluarse como true para dos expresiones de esta sentencia:puntuacion >= 70 y puntuacion >= 60.

Sin embargo, en el momento de ejecución, el sistema procesa una sentencia if compuesta como una sola; una vez que se ha satisfecho una condición (76 >= 70), se ejecuta la sentencia apropiada (nota = "Bien";), y el control sale fuera de la sentencia if sin evaluar las condiciones restantes.

SENTENCIA SWITCH

La sentencia switch se utiliza para realizar sentencias condicionalmente basadas en alguna expresión. Por ejemplo, un programa contiene un entero llamado “mes” cuyo valor indica el mes en alguna fecha. También requiere mostrar el nombre del mes basándose en su número entero equivalente. Podrías utilizar la sentencia switch de Java para realizar esta tarea.

```
int mes;
. . .
switch (mes) {
case 1:  System.out.println("Enero"); break;
case 2:  System.out.println("Febrero"); break;
case 3:  System.out.println("Marzo"); break;
case 4:  System.out.println("Abril"); break;
case 5:  System.out.println("May0"); break;
case 6:  System.out.println("Junio"); break;
case 7:  System.out.println("Julio"); break;
case 8:  System.out.println("Agosto"); break;
case 9:  System.out.println("Septiembre"); break;
case 10: System.out.println("Octubre"); break;
case 11: System.out.println("Noviembre"); break;
case 12: System.out.println("Diciembre"); break;
}
```

La sentencia switch evalúa su expresión, en este caso el valor de mes, y ejecuta la sentencia case apropiada.

Decidir cuando utilizar las sentencias if o switch dependerá del juicio personal. Se puede decidir, cual utilizar basándose en la buena lectura del código o en otros factores.

Cada sentencia case debe ser única y el valor proporcionado a cada sentencia case debe ser del mismo tipo que el tipo de dato devuelto por la expresión proporcionada a la sentencia switch.

Otro punto de interés en la sentencia switch son las sentencias break después de cada case. La sentencia break hace que el control salga de la sentencia switch y continúe con la siguiente línea. La sentencia break es necesaria porque las sentencias case se siguen ejecutando hacia abajo. Esto es, sin un break explícito, el flujo de control seguiría secuencialmente a través de las sentencias case siguientes.

En el ejemplo anterior, no se quiere que el flujo vaya de una sentencia case a otra, por eso se utilizaron las sentencias break.

Sin embargo, hay ciertos escenario en los que se requiere que el control proceda secuencialmente a través de las sentencias case. Como este código que calcula el número de días de un mes.

```
int mes;
int numeroDias;
switch (mes) {
case 1.
case 3.
case 5.
case 7.
case 8.
case 10.
case 12.
   numeroDias = 31;
   break;
case 4.
case 6.
case 9.
case 11.
   numeroDias = 30;
   break;
case 2.
   if ( ((ano % 4 == 0) && !(ano % 100 == 0)) || ano % 400 == 0) )
      numeroDias = 29;
   else
      numeroDias = 28;
   break;
}
```

Finalmente, se puede utilizar la sentencia default al final de la sentencia switch para manejar los valores que no se han manejado explícitamente por una de las sentencias case.

```
int mes;
. . .
switch (mes) {
case 1:  System.out.println("Enero"); break;
case 2:  System.out.println("Febrero"); break;
case 3:  System.out.println("Marzo"); break;
case 4:  System.out.println("Abril"); break;
case 5:  System.out.println("Mayo"); break;
case 6:  System.out.println("Junio"); break;
case 7:  System.out.println("Julio"); break;
case 8:  System.out.println("Agosto"); break;
case 9:  System.out.println("Septiembre"); break;
case 10: System.out.println("Octubre"); break;
```

```
case 11: System.out.println("Noviembre"); break;
case 12: System.out.println("Diciembre"); break;
default: System.out.println("Ese, no es un mes válido!");
      break;
  }
```

SENTENCIAS WHILE.

Generalmente hablando, una sentencia while realiza una acción “mientras” se cumpla una cierta condición. La sintaxis general de la sentencia while es.

```
while (expresión)
   sentencia
```

Esto es, mientras la expresión sea verdadera, ejecutará la sentencia.

Sentencia puede ser una sola sentencia o puede ser un bloque de sentencias. Un bloque de sentencias es un juego de sentencias legales de java contenidas dentro de corchetes ('{'y '}').

Por ejemplo, que además de incrementar contador dentro de un bucle while también se requiere imprimir el contador cada vez que se lea un carácter. Podrías escribir esto en su lugar.

```
while (System.in.read() != -1) {
   contador++;
   System.out.println("Se ha leído un el carácter = " + contador);
}
```

Además de while Java tiene otros dos constructores de bucles que puedes utilizar en tus programas.

Sentencia FOR y DO-WHILE.

Primero el bucle for. Puedes utilizar este bucle cuando conozcas los límites del bucle (su instrucción de inicialización, su criterio de terminación y su instrucción de incremento). Por ejemplo, el bucle for se utiliza frecuentemente para iterar sobre los elementos de un array, o los caracteres de una cadena.

```
// a es un array de cualquier tipo

int i;
int length = a.length;
for (i = 0; i < length; i++) {
   // hace algo en el elemento i del array a
}
```

Si se sabe cuando se esta escribiendo el programa que se quiere empezar en el inicio del array, parar al final y utilizar cada uno de los elementos. Entonces la sentencia for es una buena elección. La forma general del bucle for puede expresarse así.

```
for (inicialización; terminación; incremento)
   sentencias
```

Inicialización es la sentencia que inicializa el bucle se ejecuta una vez al iniciar el bucle.

Terminación es una sentencia que determina cuando se termina el bucle. Esta expresión se evalúa al principio de cada iteración en el bucle. Cuando la expresión se evalúa a false el bucle se termina.

Finalmente, incremento es una expresión que se invoca en cada interacción del bucle. Cualquiera (o todos) de estos componentes pueden ser una sentencia vacía (un punto y coma).

Java proporciona otro bucle, el bucle do-while, que es similar al bucle while que se vio al principio, excepto en que la expresión se evalúa al final del bucle.

```
do {
   sentencias
} while (Expresión Booleana);
```

La sentencia do-while se usa muy poco en la construcción de bucles pero tiene sus usos. Por ejemplo, es conveniente utilizar la sentencia do-while cuando el bucle debe ejecutarse al menos una vez. Por ejemplo, para leer información de un fichero, sabemos que al menos debe leer un carácter.

```
int c;
InputStream in;
. . .
do {
   c = in.read();
   . . .
} while (c != -1);
```

4.3.5 Introducción al paquete Java.lang

IMPORT JAVA.LANG.SYSTEM.

La clase System, a su vez, contendrá en su interior una import java.io (que es el paquete informático para la entrada y la salida) para acceder a las clases de la entrada y de la salida, y las usará para mandar lo que queramos en la pantalla.

El paquete informático java.lang. es el más importante de Java, en éste están contenidos las clases fundamentales del lenguaje, hasta el punto de que no hace falta declarara el import, porque Java lo introduce automáticamente.

Además me gustaría detenerme en un aspecto fundamental de la introducción de los paquetes informáticos. Si importo en mi archivo el paquete informático java.lang, éste importará a su vez el paquete informático java.io, y yo desde mi archivo no podré usar las clases de java.io; para hacerlo tengo que importarlo explícitamente. Esto sucede aunque programe una aplicación en más de un archivo (con más clases), en cada archivo tengo que importar los paquetes informáticos que necesito para la clase que estoy estableciendo, no basta con importarlos en una sola.

Las clases que contiene este paquete informático java.lang tan importante para el lenguaje son:

BOOLEAN: Los tipos de datos se pueden representar también con objetos dichos;

CONTENEDORES: es una clase que genera los contenedores para los tipos de datos boolean (verdadero, falso).

BYTE: es la clase que genera los contenedores para los tipos de datos enteros cortos (de un byte).

CARÁCTER: es una clase que genera los contenedores para los tipos de datos caracteres.

CHARACTER.SUBSET: es la clase que genera los contenedores para los tipos de datos caracteres con codificación unicode.
CHARACTER.UNICODEBLOCK: es la clase que genera los contenedores para los tipos de caracteres con codificación unicode 2.0.

CLASS: representa las clases y las interfaces a runtime (en ejecución), en Java es posible, en tiempo de ejecución, crear, ejecutar y compilar clases. El compilador y el ejecutor se mezclan de una forma rara, sin embargo a nosostros no nos sirven estas clases "especiales" porque escribiremos unas aplicaciones fáciles.

CLASSLOADER: cargador de clases a tiempo de ejecución.

COMPILER: compilador de clases a tiempo de ejecución.

DOUBLE: es la clase que genera los contenedores para los tipos de datos reales en coma móvil de 64 bit.

FLOTA: es la clase que genera los contenedores para los tipos de datos reales de 32 bit en coma móvil.

INTEGER: es la clase que genera los contenedores para los tipos de datos enteros.

LONG: es la clase que genera los contenedores para los tipos de datos enteros dobles.

MATH: contiene métodos que simulan funciones matemáticas.

NUMBER: clase de objetos contenedores de números genéricos.

OBJECT: es la clase de la que derivan todas las clases del lenguaje Java, es decir, cada clase es subclase (no necesariamente directa) de ésta.

PACKAGE: contiene métodos para extrapolar informaciones en los paquetes informáticos de Java.

PROCESS: Java es un lenguaje que permite gestionar Thread, es decir pequeños programas que corren en paralelo. Esta clase se ocupa de ellos, como las demás clases de java.lang: Runtime, RuntimePermission, Thread, ThreadGroup, ThreadLocal, Throwable

SECURITYMANAGER: para la seguridad

SHORT: es la clase que genera los contenedores para los tipos de datos enteros cortos.

STRING: es la clase que genera los contenedores para los tipos de datos cadenas de caracteres.

STRINGBUFFER: es la clase que genera los contenedores para los tipos de datos cadenas de caracteres modificables.

SYSTEM: es la clase que interactúa con el sistema y contiene muchos métodos útiles, y unas referencias a otras clases (llamadas class field), es la que usaremos, por ejemplo, para crear una salida sobre la pantalla a caracteres.

VOID: es la clase que genera los contenedores para los tipos de datos void, es decir, sin tipo. Éste parece un tipo de dato inútil, sin embargo veremos que no lo es en absoluto, además es utilísimo. En realidad, es útil cuando queremos definir un método-procedimiento (un método que no tiene valor de vuelta), en En Java existen sólo dos funciones que obligan a devolver un valor del tipo que declaran. Declarando un método como void, se simula el procedimiento, es decir, no se necesita la vuelta.

PAQUETES DE APLICACIÓN DE JAVA (API. APLICATION PROGRAMING INTERFAZ).

En Java, es posible agrupar varias clases en una estructura llamada paquete. Un paquete no es más que un conjunto de clases, generalmente relacionadas entre sí de alguna manera. Es habitual diseñar una aplicación distribuyendo su funcionalidad entre varios paquetes, cuyas clases se comunican entre sí a través de interfaces bien definidas.

El uso de paquetes aporta varias ventajas frente a la programación sin paquetes. En primer lugar, permite encapsular funcionalidad en unidades con un cierto grado de independencia, ocultando los detalles de implementación. De esta forma se pueden conseguir diseños e implementaciones más limpios y elegantes.

Por otra parte, se potencia la reutilización de las clases desarrolladas. Es posible definir interfaces de uso de cada paquete, para que otros paquetes o aplicaciones puedan utilizar la funcionalidad implementada.

Además, el uso de paquetes permite la reutilización de los nombres de las clases, ya que el espacio de nombres de un paquete es independiente del de otros paquetes. El lenguaje Java impone la restricción de que una clase debe tener un nombre único, dentro del paquete al cual pertenece. Sin embargo, es posible que dos clases tengan el mismo nombre, siempre y cuando pertenezcan a paquetes distintos.

El lenguaje Java proporciona una serie de paquetes que incluyen ventanas, utilidades, un sistema de entrada/salida general, herramientas y comunicaciones. En la versión actual del JDK, los paquetes Java que se incluyen son:

java.applet. este paquete contiene clases diseñadas para usar con applets en los Web Browsers. Hay una clase Applet y tres interfaces: AppletContext, AppletStub y AudioClip.

java.awt. este paquete y sus subpaquetes implementan las clases para, a su vez, implementar los controles GUI(Interfaz Gráfico de Usuario), para implementar interfaces gráficas, además de instrumentos para el dibujo, manipulación de las imágenes, imprimir y otras funciones El paquete Abstract Windowing Toolkit (awt) incluye las clases Button, Checkbox, Choice, Component, Graphics, Menu, Panel, TextArea y TextField.

java.io. El paquete de entrada/salida contiene las clases de acceso a ficheros: FileInputStream y FileOutputStream. Los cuales permiten introducir y producir datos.

java.lang. Este paquete incluye las clases del lenguaje Java básicas, propiamente dicho: Object, Thread, Exception, System, Integer, Float, Math, String, etc.

java.net. Este paquete da soporte a las conexiones del protocolo TCP/IP y, además, incluye las clases Socket, URL y URLConnection. Permiten trabajar en red, intercomunicándose ya sea por Internet o redes locales.

java.util. Este paquete es una miscelánea de clases útiles para muchas cosas en programación. Se incluyen, entre otras, Date (fecha), Dictionary (diccionario), Random (números aleatorios) y Stack (pila FIFO).

4.3.6 La interfaz de usuario AWT.

Las aplicaciones Applets de Java son muchas y muy variadas, se pueden aplicar en diferentes áreas no solo de informática, sino con finalidades específicas, o para apoyo para la enseñanza de alguna materia como física, matemáticas, lógica u otras. Java es un lenguaje de programación por lo tanto cuanta con todas las características que permiten desarrollar cualquier tipo de programa. Se puede utilizar para hacer programas muy sencillos, hasta aplicaciones mas completas, mas adelante se verán algunos ejemplos de applets mas completos, por el momento comenzaremos con ejemplos sencillos para poder entender mejor que es un applet.

4.3.7 HTML y los applets de Java. Diseño y programación de applets.

APPLETS

Un applet es una mini-aplicación, escrita en Java, que se ejecuta en un browser(Netscape Navigator, Microsoft Internet Explorer,) al cargar una página HTML que incluye información sobre el applet a ejecutar por medio de las tags

<APPLET>... </APPLET>.

A continuación se detallan algunas características de las applets:

1. Los ficheros de Java compilados (*.class) se descargan a través de la red desde un servidor de Web o servidor HTTP hasta el browser en cuya Java Virtual Machine se ejecutan. Pueden incluir también ficheros de imágenes y sonido.

2. Las applets no tienen ventana propia: se ejecutan en la ventana del browser (en un “panel”).

3. Por la propia naturaleza “abierta” de Internet, las applets tienen importantes restricciones de seguridad, que se comprueban al llegar al browser: sólo pueden leer y escribir ficheros en el servidor del que han venido, sólo pueden acceder a una limitada información sobre el ordenador en el que se están ejecutando, etc. Con ciertas condiciones, las applets “de confianza” (trusted applets) pueden pasar por encima de estas restricciones. Aunque su entorno de ejecución es un browser, los applets se pueden probar sin necesidad de browser con la aplicación appletviewer del JDK de Sun.

Por tanto, no necesita preocuparse por un método main() ni en dónde se realizan las llamadas. El applet asume que el código se está ejecutando desde dentro de un navegador. El appletviewer se asemeja al mínimo navegador. Espera como argumento el nombre del fichero html que debe cargar, no se le puede pasar directamente un programa Java. Este fichero html debe contener una marca que especifica el código que cargará el appletviewe.

```
<HTML>
<APPLET CODE=HolaMundo.class WIDTH=300 HEIGHT=100>
</APPLET>
</HTML>
```

El appletviewer crear un espacio de navegación, incluyendo un área gráfica, donde se ejecutará el applet, entonces llamará a la clase applet apropiada. En el ejemplo anterior, el appletviewer cargará una clase de nombre HolaMundo y le permitirá trabajar en su espacio gráfico.

ALGUNAS CARACTERÍSTICAS DE LOS APPLETS

Las características de las applets se pueden considerar desde el punto de vista del programador y desde el del usuario. En este manual lo más importante es el punto de vista del programador:

- Las applets no tienen un método main() con el que comience la ejecución. El papel central de su ejecución lo asumen otros métodos que se verán posteriormente.
- Todas las applets derivan de la clase java.applet.Applet. Las applets deben redefinir ciertos métodos heredados de Applet que controlan su ejecución: init(), start(), stop(), destroy().
- Se heredan otros muchos métodos de las super-clases de applet que tienen que ver con la generación de interfaces gráficas de usuario (AWT). Así, los métodos gráficos se heredan de Component, mientras que la capacidad de añadir componentes de interface de usuario se hereda de Container y de Panel.
- Las applets también suelen redefinir ciertos métodos gráficos: los más importantes son paint() y update(), heredados de Component y de Container; y repaint() heredado de Component
- Las applets disponen de métodos relacionados con la obtención de información, como por ejemplo: getAppletInfo(), getAppletContext(), getParameterInfo(),getParameter(), getCodeBase(), getDocumentBase(), e isActive().

CICLO DE VIDA DE UN APPLET

Cuando un applet se carga en el appletviewer, comienza su ciclo de vida, que pasaría por las siguientes fases como se muestra en la Figura 4.2:

- Se crea una instancia de la clase que controla el applet. En el ejemplo de la figura anterior, sería la clase HolaMundo.
- El applet se inicializa.
- El applet comienza a ejecutarse.
- El applet empieza a recibir llamadas. Primero recibe una llamada init (inicializar), seguida de un mensaje start (empezar) y paint (pintar). Estas llamadas pueden ser recibidas asíncronamente.

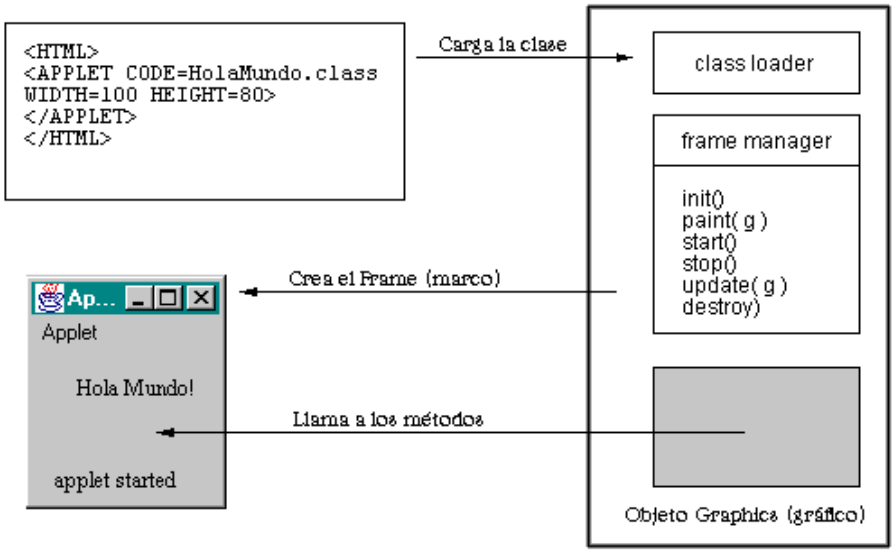


Figura 4.2. Ciclo de vida de un Applet.

4.4 Aplicaciones Generales

Hasta este momento se han visto las características principales con las que cuenta este leguaje de programación, a continuación mostraremos como se puede trabajar con el y que software es el que se requiere para su buen funcionamiento de Java.

J2SE (Java 2 Standard Edition)

Es el conjunto de herramientas de software que permite el desarrollo y la ejecución de programas Java destinados al lado cliente. Antes se le llamaba JDK (Java Development Kit) o Kit de Desarrollo de Programas Java.

Es gratuito y de libre distribución. Se puede descargar desde la página oficial de Sun Microsystems relacionada con Java http://java.sun.com/.

Sun Mycrosystems es la empresa americana que creó el lenguaje de programación Java allá por el año 1995. Se promocionan diciendo que son el punto en las empresas puntocom y abogan por escribir código que cumpla la premisa "Write Once, Run Anywhere" que podría traducirse como "Escribe código una vez donde tu quieras y ejecútalo cuantas veces quieras donde tú quieras, sin ningún retoque ni recompilación".

Dentro del J2SE se incluyen el compilador y la JVM (Java Virtual Machine) o Máquina virtual de Java. También se la conoce como Intérprete de Java.

Cada plataforma tiene su propia versión. En la página de Java dentro de Sun puede descargarse el J2SE para Windows, Linux, Solaris, etc.

Instalación de la JDK (Maquina Virtual de Java).

A continuación se trabajara con la instalación de la JDK, SE que se bajo de la pagina de SUN, se lograra la instalación completa y se comprobara ejecutando un pequeño programa en Java.

El SDK (Kit de Desarrollo de Java) o JDK es un conjunto de herramientas y utilerías que en resumen son:

javac El compilador Java por excelencia, un compilador de línea de comandos, que te permitirá crear tus programas y applets en Java.

appletviewer Un visualizador de Applets para no tener que cargarlos en un navegador.

java El intérprete que te permitirá ejecutar tus aplicaciones creadas en Java.

javadoc El documentador de Java

jdb El depurador de Java

javap Un descompilador que te permite ver el contenido de las clases compiladas.

Para Instalar el Java 2 SDK en Windows haga doble clic en el archivo de instalación. Es importante que instale todo el SDK, tanto programas como documentación desde la carpeta (directorio) raíz, C:\ u otra unidad como la D:\ o la E:\ . En el cuadro de diálogo, se pregunta si desea instalar el SDK, SE 1.4.1_02, se despliega el Asistente de configuración del SDK como se muestra en la Figura 4.3

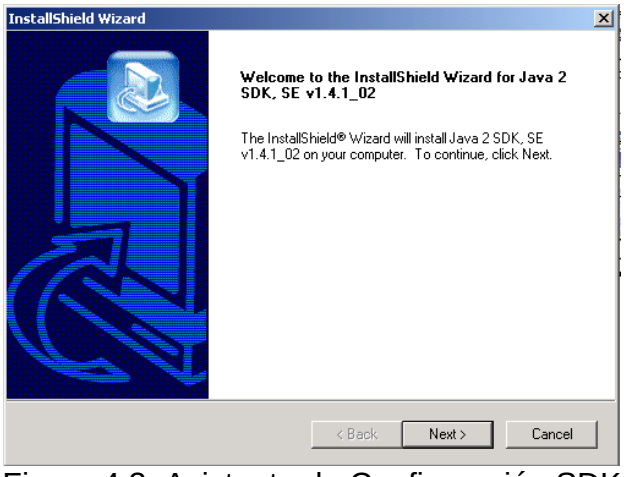


Figura 4.3. Asistente de Configuración SDK

El asistente instalará tres componentes del SDK:

- Archivos de programa. Son los programas ejecutables necesarios para crear, compilar y verificar el funcionamiento de sus proyectos de Java.
- Archivos de biblioteca y encabezados. Archivos usados únicamente por los programadores que hacen llamadas a código nativo desde programas de Java.
- Archivos de demostración. Son programas de Java 2, con versiones que puede ejecutar y archivos fuente que puede examinar para aprender más acerca del lenguaje.
- Biblioteca de clases o API's (Application Program Interface) . Que son las librerías de clases llamadas paquetes creadas por los desarrolladores del software de Java de la empresa Sun.

Después de haber instalado SDK, notará que hay varios archivos instalados en el subdirectorio \J2SDK141\lib la mayoría con extensión .jar. Aunque son archivos .jar, no debe descomprimirlos. El SDK puede leer los archivos .jar en su formato de archivo en este directorio.

En este caso se puede escoger la unidad destino donde se va a instalar el software de Java 2 SDK SE dándole clic al botón Browse... Para este ejemplo se seleccionó la unidad C: y aparece como c:\j2sdk1.4.1_02. (Figura 4.4).

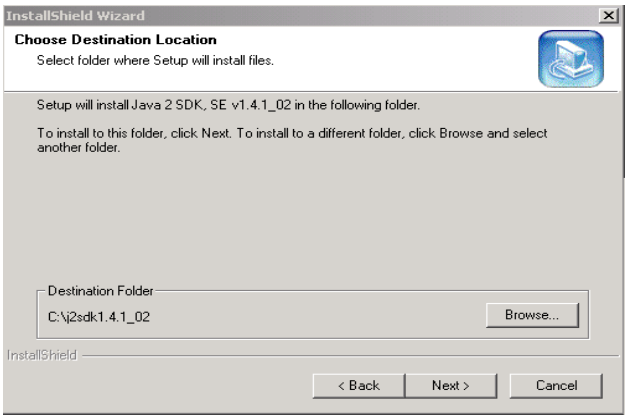


Figura 4.4. Selección de Unidad destino de instalación

CONFIGURACIÓN DE LAS VARIABLES DE ENTORNO PARA JAVA.

Son dos variables de ambiente del sistema operativo Windows que tienen que ser configuradas, estas son PATH y CLASSPATH.

CONFIGURACIÓN DE LA VARIABLE DE AMBIENTE PATH.

La variable de ambiente PATH indica al sistema operativo donde se ubican o se encuentran los programas ejecutables del kit de herramientas del Java 2 SDK SE, en la documentación indica que debe apuntar a la carpeta bin, que es donde se alojan los archivos ejecutables, así por ejemplo si instalaste el Java en el disco duro C: se tiene que poner:

SET PATH = C:\J2SDK141\bin
Entonces el sistema operativo sabrá donde buscar esos archivos.

Es recomendable establecer otra variable de ambiente conocida como JAVA_HOME que apunta a donde instalaste el Java , así:

JAVA_HOME = C:\J2SDK141

Entonces puedes establecer tu variable de ambiente PATH, así:

SET PATH =%JAVA_HOME%\bin

lo cual resulta mas práctico a la larga cuando instalas mas software que trabaje con Java como el servidor de Servlets y JSP Tomcat de Apache.

CONFIGURACIÓN DE LA VARIABLE DE AMBIENTE CLASSPATH

La variable CLASSPATH indica al compilador, e interprete de Java y a otras aplicaciones que utilicen las API's de Java donde ubicarlas o encontrarlas para cargarlas a Memoria y utilizarlas. En la versión Java 2 SDK 1.4.1 las API's están en la carpeta lib en formato .jar (Java Archive) y son los archivos tools.jar y dt.jar, entonces hay que configurar la variable CLASSPATH para que apunte a esos archivos porque ahí están las clases compactadas de las API's, no basta que apunten a la carpeta, tienen que apuntar a los archivos así:

SET CLASSPATH=.;C:\J2SDK141\lib\tools.jar;C:\J2SDK141\lib\dt.jar

o si ya creaste la variable JAVA_HOME para apuntar a C:\J2SDK141 se puede hacer lo siguiente:

SET CASSPATH=.;%JAVA_HOME%\lib\tools.jar;%JAVA_HOME%\lib\dt.jar

la parte de la ruta de configuración del CLASSPATH .; (punto y punto y coma) es necesaria para apuntar a la carpeta o directorio actual de trabajo con el fin de que podamos compilar y ejecutar nuestros programas de Java en la carpeta donde estemos ubicados en ese momento y tome las clases generadas en esa carpeta.

Para establecer las variables de ambiente PATH YCLASSPATH Microsoft Windows NT, 2000, y XP, hay que seguir los pasos siguientes

Inicio->Panel de Control ->Sistema ->Ventana de "Propiedades del Sistema"; ficha o pestaña "Avanzado"; botón "Variables de entorno" ->Ventana de "Variables de entorno".

En la Figura 4.5 nos muestra la ventana de Variable de entrono ahí se encuentran las variables de usuario en la parte superior, pulsando el botón "Nueva" aparece un cuadro de dialogo donde se debe introducir el Nombre de la variable y el Valor de la Variable. Esto es en el caso de Windows XP profesional:

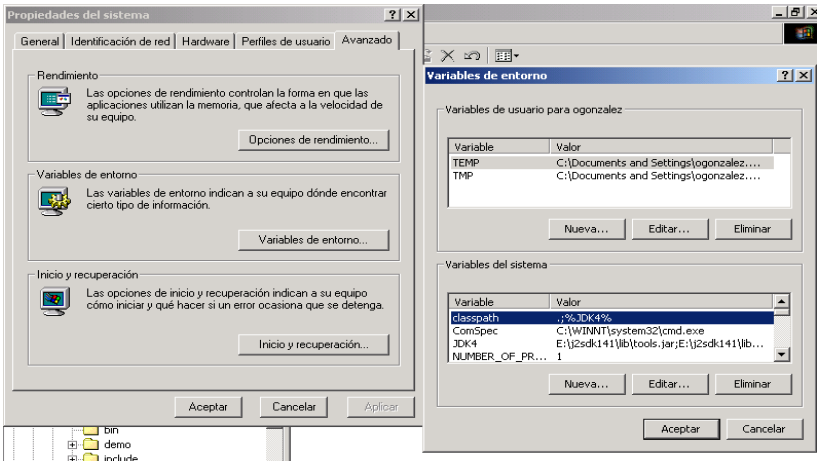


Figura 4.5. Variables de Entorno.

Pruebas de Instalación

Para lograr hacer una prueba de la instalación los usuarios de Windows pueden verificar su instalación del JDK al usar el comando de MS-DOS en la mayoría de los sistemas.

Se debe escribir lo siguiente en un indicador de comandos para verificar que su sistema pueda encontrar la versión correcta del JDK en él:

Java–version

Si está usando el JDK 1.4.1_02, en respuesta se debería ver el siguiente mensaje como se muestra en la Figura 4.6:

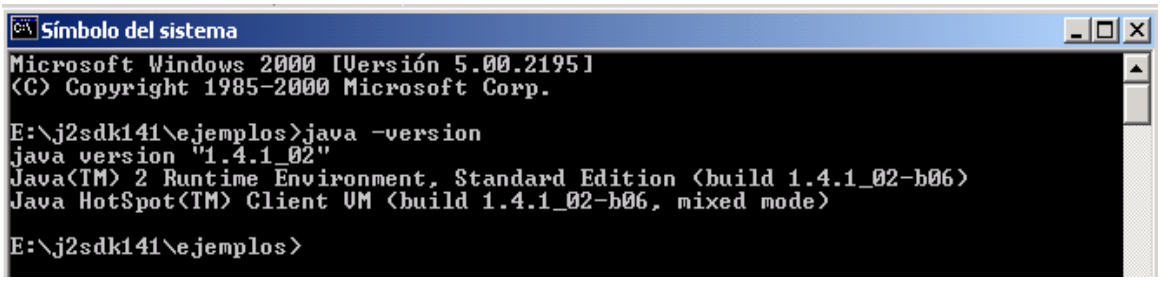


Figura 4.6. Versión de java

SU PRIMERA APLICACIÓN DE JAVA

Las aplicaciones de Java son programas“independientes” que no requieren un navegador Web para correr. Son más parecidos a los programas que se suelen usar en la computadora (los ejecuta localmente con su ratón o escribiendo el nombre en la línea de comandos)

Puede utilizar el block de notas de Windows o cualquier editor de texto que conozca para escribir el programa.

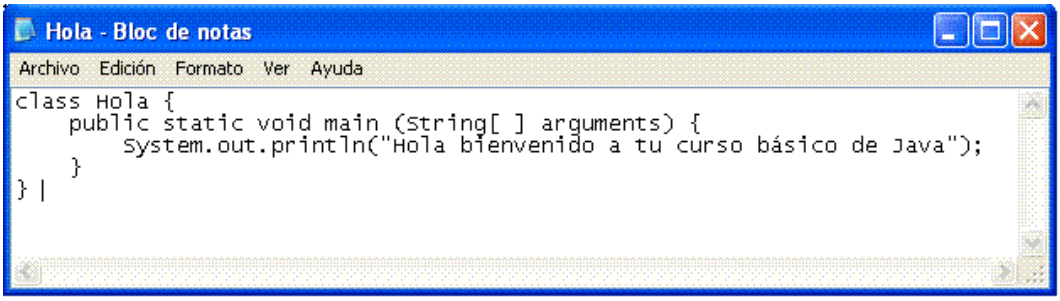


Figura 4.7. Archivo Hola.java

Se creara una carpeta en c:cursoj ahí se debera guardar el archivo del programa con el nombre de Hola.java (Figura 4.7). Es importante que el programa se llame exactamente igual al nombre de la clase para que pueda compilarse, de lo contrario habrá un error. Cuando guarde el archivo utilice comillas “ “ antes y después del nombre para evitar que el programa guarde el archivo con su extensión .TXT

COMPILACIÓN Y EJECUCIÓN DEL PROGRAMA EN WINDOWS

Se debe cambiar el directorio actual al cursoj con la ventana de una sesión de MS-DOS con el comando:
cd\cursoj

Si se encuentra en la carpeta correcta, puede compilar Hola.java escribiendo lo siguiente en el indicador de línea de comandos:

Javac Hola.java
Si el compilador del JDK no despliega ningún mensaje de error quiere decir que se compilo con éxito. Esto quiere decir que se creará un archivo Hola.class en el mismo directorio que contiene Hola.java.
Este archivo .class es el código de bytes (byte code) que puede ser ejecutado por la máquina virtual

Una vez que tenga un archivo .class, lo puede ejecutar mediante el intérprete de código de bytes. Ejecute Hola.class escribiendo lo suguiente:

Java Hola

En este caso se va a desplegar la leyenda que esta en el programa que dice “Bienvenido a tu curso básico de Java”.

Hasta aquí hemos logrado que la Maquina Virtual de Java funcione correctamente siguiendo los pasos de configuración anteriores, pero tambien se necesita alguna software que nos ayude a hacer un poco mas sencillos nuestros programas, en este caso, el software que vamos a utilizar es el IDE de Kawa para Java

4.5 Conclusiones

Java es muy utilizado para escribir programas que trabajen en Internet, empezando por las bases hasta llegar a las interfaces gráficas los cuales son los aspectos más avanzados de la programación con Java. Con estas bases se puede lograr que algunos desarrollen sus propias aplicaciones, aunque no sean muy avanzadas pero con la práctica y con la investigación se lograra algo mejor.

En conclusión listaremos algunas de las ventajas que se tienen al programar con Java:

Primero: No se requiere volver a escribir el código si se quiere ejecutar el programa en otra máquina. Un solo código funciona para todos los browsers compatibles con Java o donde se tenga una Máquina Virtual de Java (Mac's, PC's, Sun's, etc).

Segundo: Java es un lenguaje de programación orientado a objetos, y tiene todos los beneficios que ofrece esta metodología de programación como son herencia, polimorfismo entre otras.

Tercero: Un browser compatible con Java deberá ejecutar cualquier programa hecho en Java, esto ahorra a los usuarios tener que estar insertando "plug-ins" y demás programas que a veces nos quitan tiempo y espacio en disco.

Cuarto: Java es un lenguaje y por lo tanto puede hacer todas las cosas que puede hacer un lenguaje de programación: Cálculos matemáticos, procesadores de palabras, bases de datos, aplicaciones gráficas, animaciones, sonido, hojas de cálculo, etc. Por lo tanto se puede desarrollar lo que sea necesario.

Quinto: Si lo que interesa son las páginas de Web, ya no tienen que ser estáticas, se le pueden poner toda clase de elementos multimedia y permiten un alto nivel de interactividad, sin tener que gastar en paquetes carísimo de multimedia.

En general Java posee muchas ventajas y se pueden hacer cosas muy interesantes con esto. Hay que prestar especial atención a lo que está sucediendo en el mundo de la computación, Java posee mucha fuerza y es tema en cualquier medio computacional. Muchas personas apuestan a futuro y piensan en Java.

CAPÍTULO 5

5. DIDÁCTICA DE LA PROGRAMACIÓN

5.1 Objetivo

Revisar las estrategias para la enseñanza y evaluación en áreas de cómputo.

5.2 Introducción

La complejidad de los programas que se desarrollan actualmente produce la necesidad de iniciar a los alumnos en un camino que los conduzca a utilizar efectivas técnicas de programación. Es importante para ello poner énfasis en el diseño previo.

Una estrategia valedera es comenzar a enseñar programación utilizando los algoritmos como recursos esquemáticos para plasmar el modelo de la resolución de un problema. Esto genera la inclusión de una etapa previa a la programación que resulta un tanto tediosa tanto para los alumnos ávidos de utilizar la computadora como para aquellos que la utilizan abitualmente.

Además, si bien no aparecen dificultades graves con el aprendizaje de estas técnicas, no resulta una tarea trivial obtener un algoritmo semánticamente correcto, sino que para lograrlo se requieren sucesivos refinamientos.

Es por ello que la educación y la instrucción de las nuevas generaciones es una labor compleja y sutil. Existen actualmente todo un conjunto de principios, criterios, normas, recursos y técnicas de acción educativa, este conjunto de doctrinas, principios, normas y técnicas de acción educativa se le conoce como didáctica de la programación.

5.3 Desarrollo

Las metodologias de la enseñanza, son dinámicas. Se definen en función de varios procesos o momentos metodológicos, no secuenciadas, e involucra en forma a todos los demás elementos de la estructura didáctica, a continuación trataremos algunos de los elementos más importantes.

5.3.1 Elementos de didáctica y manejo de grupos.

Los procesos que la constituyen son:

ESTRUCTURACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE: Es el diseño de las actividades que el profesor considera que el estudiante debe realizar en el proceso de enseñanza - aprendizaje para que ejerza la acción sobre el contenido para que se lo apropie.

DISEÑO DE RECURSOS: Es la organización y diseño de los materiales de tal manera que los estudiantes actúen con y sobre el contenido. Esto no se refiere sólo a lo que comúnmente de se denomina materiales didácticos.

Los alumnos y profesores constituyen los elementos personales del proceso, siendo un aspecto crucial, el interés y la dedicación de docentes y estudiantes en las actividades de enseñanza-aprendizaje.

Por tanto, el proceso de enseñanza – aprendizaje se desarrolla en varia etapas, y comporta un proceso de comunicación entre el docente que enseña, que transmite unos conocimientos y a quien se enseña, el alumno o también denominado discente.

Son cinco los componentes de la situación docente que la didáctica procura analizar, integrar funcionalmente y orientar para los efectos prácticos de la labor docente: el educando, el maestro, los objetivos, las asignaturas y el método.

a) El educando, no sólo como alumno que debe aprender con su memoria y con su inteligencia, sino como ser humano en evolución, con todas sus capacidades y limitaciones, peculiaridades, impulsos, intereses y reacciones, pues toda esa compleja dinámica vital condicionará su integración en el sistema cultural de la civilización.

b) El maestro, no sólo como explicador de la asignatura, sino como educador apto para desempeñar su compleja misión de estimular, orientar y dirigir con habilidad el proceso educativo y el aprendizaje de sus alumnos, con el fin de obtener un rendimiento real y positivo para los individuos y para la sociedad.

c) Los objetivos que deben ser alcanzados, progresivamente, por el trabajo armónico de maestros y educandos en las lides de la educación y del aprendizaje. Estos objetivos son la razón de ser y las metas necesarias de toda la labor escolar y deben ser el norte de toda la vida en la escuela y en el aula.

d) Las asignaturas, que incorporan y sistematizan los valores culturales, cuyos datos deberán ser seleccionados, programados y dosificados de forma que faciliten su aprendizaje, fecundando, enriqueciendo y dando valor a la inteligencia y a la personalidad de los alumnos. Las asignaturas son los reactivos culturales empleados en la educación y los medios necesarios para la formación de las generaciones nuevas.

e) El método de enseñanza, que fusiona inteligentemente todos los recursos personales y materiales disponibles para alcanzar los objetivos propuestos, con más seguridad, rapidez y eficacia. De la calidad del método empleado dependerá, en gran parte, el éxito de todo el trabajo escolar.

MANEJO DE GRUPOS

En este contexto, deben manejarse lo que se ha llamado Dinámica de grupos, Técnicas de dinámica en grupo y Técnicas didácticas.

Al hablar de aprendizaje en un sentido amplio, nos referimos a la interacción de un sujeto con un objeto. Esta interacción se da en un proceso dinámico cuyos productos son manifestables en actividades y actitudes en conductas concretas, entendida conducta como despliegue funcional del sujeto.

En situaciones escolares cuando nos referimos al proceso de aprendizaje, hablamos de la interacción alumno - contenido. En esta situación escolar se organizan a los sujetos en grupos referidos a un contenido con un cierto nivel de dificultad y con una modalidad (teorica, practica, intrumental o metodológica).

El comportamiento de sujetos - humanos en grupo, genera lo que se llama dinámica en grupo; la conducci{on de los sujetos hacia ciertas actitudesconstituye lo que se llama técnicas de dinámica de grupos; y las técnicas que conducen las actividades didácticas, es decir las que tienen por función el aprendizaje de un contenido se llaman procedimientos y técnicas didácticas.

El trabajo en grupo es un método que permite a los alumnos convenientemente agrupados, realizar y discutir un trabajo concreto, intervenir en una actividad

exterior, o encontrar solución a un problema sometido al examen del grupo, con la finalidad de concluir con unos razonamientos concretos. El trabajo en grupo permite conseguir unos objetivos distintos a los métodos expositivos, al facilitar una mayor participación y responsabilidad de los alumnos.

Los objetivos generales de este método son:

- Lograr la individualización de la enseñanza.
- Conseguir la participación activa de todos los alumnos en el proceso de enseñanza-aprendizaje.
- Desarrollar la habilidad de trabajar en equipo.
- Los grupos restringidos poseen gran capacidad autoformativa. Por medio de la dinámica de grupo se puede cambiar las actitudes de forma más fácil que actuando individualmente.

5.3.2 Motivación en cómputo.

Existen varios posibles motivos de motivación en computo, a continuación se listan algunos de ellos:

- Motivación por el contenido terminal del aprendizaje, es decir, motivación porque lo que hay que aprender por sí mismo es interesante. La importancia de los contenidos para los futuros estudios, profesión, carrera profesional, etc. No es fácil que alguien esté motivado hacia algo que desconoce, bien en sí mismo, bien en sus resultados. El profesor tiene con respecto a esta motivación una gran tarea. De su labor mostrando la importancia de la asignatura depende en buena parte la respuesta del alumno. Si además el alumno capta el entusiasmo del profesor por la asignatura, ésta es una de las fuentes de motivación más contagiosas que se conocen y ampliamente verificada de forma empírica.

- Motivación por mediación instrumental. El alumno capta la importancia de un aprendizaje como instrumento útil para el logro de un objetivo deseado.

- Motivación por el método didáctico. Los alumnos se sienten atraídos a causa de la metodología atractiva que el profesor utiliza, pero no sólo por el lado de la amenidad, sino por el lado de la participación, el desafío intelectual, el alto nivel de los procesos mentales, etc.

- Motivación por el profesor. En el contacto entre el docente y el alumno, y de cómo éste se establece, reside una poderosa razón motivadora en los procesos de enseñanza-aprendizaje. Tal como manifiestan numerosos autores y respalda la investigación. La investigación en formación ha mostrado que en orden a fomentar los mejores desempeños en los

estudiantes, se deben establecer altas expectativas, lo cuál es válido para la mayoría de los procesos de formación.

- Motivación por la experiencia del éxito. Es bien conocido, que toda experiencia de éxito representa un refuerzo psicológico motivacional para proseguir la realización de una tarea.

5.3.3 Visión del módulo y perfil de actividades.

Las metodologías de la enseñanza, son dinámicas. Se definen en función de varios procesos o momentos metodológicos, no secuenciadas, e involucra en forma a todos los demás elementos de la estructura didáctica, los procesos que los contribuyen son:

ESTRUCTURACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE.

En el diseño de las actividades que el profesor considera que el estudiante debe realizar en el proceso de enseñanza-aprendizaje para que ejerza la acción sobre el contenido para que se lo apropie.
En términos generales las actividades deben ser diseñadas para que sean experienciales, esto es:

1. Ser vividas.
2. Ser posibles
3. Estar diversificadas
4. Ser satisfactorias
5. Ser productivas

Lo que se pretende con esto, es que el alumno (sujeto al proceso de aprendizaje) actúe, es decir, realice actividades que han sido condicionadas para la apropiación del contenido, para lograr un desempeño profesional acorde con los objetivos de la carrera. Por su puesto es el profesor el que estructura, propone y dirige dichas actividades. En función de la dinámica puede modificar su propuesta.

Las técnicas didácticas son actividades tipo, cada profesor hace las adecuaciones pertinentes a la modalidad de contenidos que maneja, las características de los grupos, de los alumnos, de la infraestructura didáctica.
Las técnicas didácticas obedecen a una lógica de desarrollo de los contenidos en tres momentos lógicos de introducción, desarrollo y síntesis, si el profesor o los alumnos desean hacer modificaciones o bien sugerir actividades no tipificadas, deberán completar esta lógica.

DISEÑO DE RECURSOS.

Es la organización y diseño de los materiales de tal manera que los estudiantes actúen como y sobre el contenido. Esto no se refiere sólo a lo que comúnmente se denomina “materiales didácticos”.

Los recursos son los elementos en los que están sostenidos los contenidos de aprendizaje. Desde luego la capacidad para poder organizar los materiales dependerá de la formación, la experiencia, la creatividad y el interés del profesor y de la medida en que sepa estructurar el contenido.

5.3.4 Errores comunes en la enseñanza de la programación.

Durante la enseñanza de la programación los profesores suelen caer en algunos errores comunes, a continuación listamos algunos de ellos:

El profesor nunca se preocupa por verificar los errores o pernsar acerca de sus propios posibles errores.

- Generalmente el profesor no FORMALIZA a nivel teorico su materia.
- No se considera el nivel de conocimiento del tema de los alumnos.
- No se llevan a cabo los suficientes ejercicios que ejemplifiquen la meta de nuestro proyecto final.
- Antes de continuar al siguiente paso no se han planteado la meta, objetivo, vision y misión.
- El profesor nunca pregunta lo obvio que solo es obvio para el mismo.

5.3.5 Técnicas de enseñanza de la programación.

TÉCNICAS DINÁMICAS.

Al hablar de aprendizaje es un sentido amplio, nos referimos a la interacción de un sujeto con un objeto. Esta interacción se da en un proceso dinámico cuyos

productos son manifestarles en actividades y actitudes, en conductas concretas, entendida conducta como despliegue funcional del sujeto.

Dinámica de grupo.

Es la interacción que se manifiesta en un grupo, surgida de las relaciones entre sus miembros. Las “fuerzas” que relacionan a los miembros de un grupo son: sus intereses, habilidades, hábitos, tendencias, frustraciones, entre otras, es decir, sus comportamientos y actitudes. Todo esto da origen a la “didactica” del grupo.

Se le llama “dinámica” porque es un término que consta de movimiento, acción, cambio, reacción, trasformación, etc.

Las técnicas dinámicas de grupo. Son procedimientos que se emplean con el fin de dirigir la acción de un grupo hacia el comportamiento de un objetivo. Estos procedimientos han sido estructurados en situaciones experimentales o controladas, fundados en principios de la psicología social y probados en la práctica, de tal manera que cuando se usan apropiadamente dan resultados similares a los obtenidos en la situación experimental. Estas técnicas activan las motivaciones, comportamientos, actitudes individuales y estimulan las fuerzas personales.

La aplicación de estos principios a las situaciones escolares, llevo a las técnicas dinámicas de grupos y estas han requerido de una rigurosa formación para trabajalas adecuadamente y con los cuidados necesarios.

TÉCNICAS DIDÁCTICAS.

Son procedimientos mediante los cuales se organizan actividades o secuencias de actividades que llevan a la realización afectiva del proceso de enseñanza-aprendizaje.

Las técnicas didácticas son elementos de las metodologías de la enseñanza efectivizadas en un aula. Existen actividades de los alumnos que no pueden ser “metidas” en una técnica. Por otro lado, es habilidad del docente y profesional del profesor donde radica lo fundamental del éxito de la enseñanza.

Dentro de las metodologías de la enseñanza, el segundo momento a considerar son las actividades (acciones del alumno sobre el contenido), estas actividades tipificadas es a lo que llamamos técnicas didácticas. Las actividades incluidas en un programa deben de ser experienciales, para ellos a de atender a los cinco principios lógicos: ser vividas, ser posibles, estar diversificadas, ser satisfactorias, ser productivas.

Es importante, por lo mismo tener en mente varios puntos:

- Las técnicas son modos, maneras de hacer las cosas, como tales dependen de qué se hace y para qué se hace. Que se aprende o enseña y para qué . el valor que tienen lo adquieren en el uso que se les de y depende de la habilidad con la que se manejen.
- Admiten ser adoptadas a situaciones diversas, el profesor debe basarse en los principios didácticos y en su creatividad.
- Debe existir un ambiente lo suficientemente flexible para el éxito de las actividades. La imagen del profesor su comportamiento y expectativas son factores que influyen en el proceso de enseñanza-aprendizaje.

El profesor en la institución educativa aparece como responsable directo de la organización del proceso didáctico que ocurre en el salón de clase, las técnicas didácticas que suelen utilizar algunos profesores se exponen a continuación:

EXPOSITIVAS

EXPOSICIÓN. Es un discurso informal de un tema o parte de un tema, realizado por el profesor. Se usa cuando se requiere dar información para iniciar una actividad intelectual.

Ventajas:

- Se adapta a cualquier contenido.
- Permite presentar mucha información en un tiempo corto.
- Es útil con grupos numerosos.

Recomendaciones:

- Consigne en el pizarrón los elementos más importantes.
- No la utilice como única técnica en curso.
- Ayúdese con recursos didácticos.

DEMOSTRACIÓN. Muestra prácticamente el manejo de un instrumento, la elaboración de un trazo o la realización de un experimento. Se usa cuando es necesario apreciar la manipulación de un proceso.

Ventajas:

- Se puede explicar la actividad.
- Se realiza en un ritmo normal.

Recomendaciones:

- Cuidado con los resultados, si no los domina no lo aplique.
- Procure que sus explicaciones sean claras.

- Procure que todo el grupo observe lo que hace.

CONFERENCIA. Es un discurso formal de un tema por un maestro o persona especializada en el asunto a informar. Se usa cuando se desea presentar información directa y compleja al grupo.

Ventajas:

- Enfatiza ideas importantes difíciles de percibir en el texto.
- Permite economizar el tiempo.

Recomendaciones:

- No pierda de vista los objetivos de la conferencia.
- Pruebe y revise el material de apoyo (notas, sonido, cartelones, etc.)

INTERROGATIVAS.

EXPOSICIÓN CON PREGUNTAS. Es una plática que dirige un profesor o instructor, a un grupo de estudiantes. El profesor transmite información al grupo, acerca de un tema preparado previamente. Se provoca la participación de los alumnos durante la clase a través de cuestionamiento.

Recomendaciones:

Conducir la exposición de manera que los alumnos participen con preguntas al expositor.

INTERROGATORIO. Consiste en una serie de preguntas estructuradas lógicamente y con claridad. Se usa cuando se quiere obtener información, puntos de vista y la capacidad de razonamiento de los alumnos.

Ventajas:

- Despierta y conserva el interés.
- Ayuda a conocer la experiencia de los alumnos, capacidad y criterio.
- Puede durar de entre 10 y 60 min.

Recomendaciones:

- Se debe dirigir la pregunta a toda la clase para que todos sean considerados a responder.
- Evite el dialogo
- Busque preguntas que lleven a el objetivo buscado y al análisis.

DIRIGIDAS.

CORRILLOS. Es un procedimiento rápido para poner opiniones en común, en un ambiente informal, descompone un grupo numeroso en unidades pequeñas. Se usa cuando se quiere que todos los miembros del grupo externen su opinión.

Ventajas:

- Es relativamente rápida entre 10 y 20 min.
- Puede incrementar el interés de los alumnos.
- Propicia el análisis.

Recomendaciones:

- Prepare bien las indicaciones sobre el tema a tratar.
- Si no se conoce bien el tema, no use la técnica.

PHILLIPS 6'6. Es un procedimiento rápido, 6 minutos, 6 personas, para poner opiniones en común. Sus ventajas y recomendaciones son muy parecidas a las dos anteriores.

LLUVIA DE IDEAS. Es un procedimiento en que los alumnos, expresan lo primero que les viene a la mente, ya sea razonable o extravagante. Se usa cuando se necesitan ideas, se requiere estimular la imaginación y se buscan soluciones.

Ventajas:

- Centra la atención en un problema.
- Despierta el interes.
- Puede servir para hacer repasos o conexiones entre temas.

Recomendaciones:

- Debe darse en una situación en la que la gente se sienta para expresar lo que piensa.
- Defina claramente el problema o plantee bien la pregunta.
- Anote las respuestas.

ESTUDIO DIRIGIDO

LECTURA COMENTADA. Es una discusión o exposición centrada sobre la lectura de un texto escogido. Para aclarar aspectos importantes del curso.

Ventajas:

- Puede desarrollar en los alumnos la capacidad de análisis crítico.
- Ayuda a enriquecer una discusión.

Recomendaciones:

- Propicie que los alumnos participen.
- No se use muy frecuente, ya que es sumamente aburrido.

TUTORÍA. Es la relación entre un maestro y un alumno. El maestro analiza las necesidades del estudiante y le proporciona una enseñanza individualizada según la capacidad y personalidad del estudiante.

Recomendaciones:

- Evita la dependencia del alumno al permitirle participar en la dirección de su aprendizaje.
- Evitar el individualismo del alumno.

A continuación se mostraran algunas recomendaciones para lograr una buena motivación a los alumnos en el ámbito de computo, es importante tomarlas en cuenta para lograr resultados exitosos, a lo largo de su proceso como profesor de asignatura. Son algunos tips y trucos para introducir correctamente a los alumnos a la materia de programación de computadoras sin complicaciones.

Recomendaciones:

- Construya metodologías no programas.
- Enseñe sintaxis no programas.
- Enseñe de lo básico a lo complejo.
- Proporcione poder al alumno, no cree frustración
- Use su espacio, module la voz y muevase.
- Haga participar a los alumnos.
- Haga uso de los ejercicios y tareas
- Muestre al menos dos ejercicios sobre cada concepto.
- Enloquece en el concepto.
- Haga un ejercicio integrador de conceptos.
- Haga uso de pruebas de pizarrón.
- Exponga un solo concepto a la vez.
- Involucre a los alumnos en la prueba

Apoyos al autoaprendizaje.

- Muestre como usar las herramientas de depuración.
- Si la herramienta no tiene depurador, muestre como depurar.
- Muestre como usar la ayuda
- Proporcione bibliografía actual, o al menos la que exista en la biblioteca.
- Proporcione ligas a sitios en Internet.
- Proporcione libros, tutoriales y manuales electrónicos.

Trabajo fuera del aula.

- Use correo electrónico para la solución de dudas.
- Organice asesorías a estudiantes avanzados fuera del aula.
- Proponga ejercicios avanzados.
- Proponga proyectos avanzados a los estudiantes avanzados.
- Cree proyectos intergeneracionales.
- Promocione concurso sobre la materia con otros grupos, o escuelas
- Investigue las novedades en Internet.

Con el uso de todas estas recomendaciones es mas sencillo logra que los alumnos se interesen en la materia y sea más interesante el desarrollo del curso. Y por otro lado, se recomienda que comience a elaborar sus propias propuestas de trabajo.

5.3.6 El proyecto final.

El proyecto final es un trabajo que abarca todos los puntos analizados durante el curso y que generalmente se pide al final de mismo.

La definición del proyecto final consta de:

- Investigación y proyecto práctico.
- Alcance
- Oportunidad
- Apoyo a los alumnos

Los requerimientos de un proyecto final son:

- Ambigueda
- Alcance de la definición
- Esquema de solución
- Aplicabilidad

Por último se recomienda que el proyecto final se ubique en el mundo real.

5.3.7 Evaluación

SISTEMAS DE EVALUACIÓN.

Existen tres formas de evaluación: heteroevaluación, autoevaluación y evaluacion mixta, según quien sea el encargado de evaluar.

HETEROEVALUACIÓN. Esta consiste en la valoración del rendimiento escolar por parte de personas distintas al propio alumno. Esta forma ha sido generalmente aceptada y se ha venido practicando desde el comienzo mismo de la escuela como institución. Esta puede ser individual y colectiva, según el profesor evalúe a cada escolar uno a uno, o al grupo de alumnos como tal.

AUTOEVALUACIÓN. Esta consiste en la valoración, por parte del propio alumno, del rendimiento educativo que ha obtenido. Puede realizarse también de modo individual y colectivo. El mayor peligro de la autoevaluación consiste en no lograr la mayor precisión, objetividad y destreza en ella, es lógico que el profesor no te va a quedar indiferente en esta tarea, sino que presenta su ayuda en este sentido.

EVALUACIÓN MIXTA. Esta tiene lugar cuando el profesor y el alumno evalúan en común las actividades o rendimiento de este. Se trata de una evaluación conjunta que tiene lugar cuando ambos analizan determinadas tareas o rendimientos. De esta forma el alumno va emitiendo sus juicios de valor sobre lo que ha hecho al profesor, quien se encarga de aceptar o reorientar dichos juicios.

TIPOS DE EVALUACIÓN.

En un programa educativo que abarque un proceso docente de cierta entidad es preciso distinguir tres estados evolutivos:

Evaluación inicial, o de diagnostico

Esta se realiza antes de dar comienzo a la actividad docente con objeto de adecuar las programaciones a las necesidades reales. La evaluación personalizada solo puede llevarse a cabo si se conoce el punto de partida de cada alumno. A grandes rasgos distinguimos los tipos de diagnostico:

DIAGNOSTICO COGNITIVO. Con el se da a conocer el perfil instructivo de los alumnos y se señalan sus lagunas y deficiencias más destacadas. Este tipo de diagnósticos se aplican:

- Al comienzo del curso escolar.

- Al comienzo de cada ciclo
- Al ingreso de los alumnos en el centro.

DIAGNOSTICO DE LAS ACTITUDES, con el de pretende conocer las posibilidades reales de los alumnos para la adquisición de los aprendizajes. Fundamentalmente son pruebas para ver la medida de la inteligencia (cociente intelectual), pero se completas con exploraciones en torno a los intereses, hábitos y destrezas, de ahí es muy recomendable que se apliquen en distintos momentos:

- Test de Inteligencia.
- Cuestionario de personalidad e intereses, para lograr que diagnóstico sea fiable.

EVALUACIÓN PROGRESIVA O CONTINÚA.

En este caso ya se tiene al alumno establecido en el nivel correspondiente y ha sido evaluado inicialmente. Ahora el diagnostico elaborado será útil para emitir un pronostico de resultados previsibles. El alumno, a partir de este momento, está sometido a un continua y progresivo estudio valorativo por parte del profesor.

Este tipo de evaluación descansa en una fijación precisa de objetivos, y en una programación de actividades coordenadas cada consecución de los mismos.

Por otro lado no hay nada mas motivador del conocimiento inmediato de los resultados, pues ello oriental alumno por el camino adecuado. Por eso mismo se ha afirmado que la evaluación progresiva es la base permanente de la planificación diaria, semana, quincenal, etc, del trabajo escolar.

No obstante, no siempre se hace buen uso de la evaluación, resultando a veces incluso deformante. Algunos profesores aplican un solo examen, o dos, al final de un largo periodo y juzgan el progreso del alumno sin tener en cuanta más datos. Como consecuencia, el alumno descuida su preparación diaria, no presta atención debida a las explicaciones y tampoco subsanan las deficiencias del aprendizaje por que no se le da la oportunidad de hacerlo.
En conclusión, para evaluar con acierto es necesario comprobar los progresos de los alumnos, no sólo al final del proceso, sino desde su principio y a lo largo del mismo, para no alejarse de la trayectoria individual y sintonizar con el ritmo de los distintos aprendizajes. La evaluación debe estar presente desde principios de la acción educadora. No es algo que surge al final para comprobar los resultados.

EVALUACIÓN DE LOS RENDIMIENTOS O GLOBAL.

Cuando se trata de conocer los resultados de un proceso tras un periodo de vigencia, se habla de evaluación de los rendimientos o global. Evaluar los rendimientos significa valorar la productividad de los alumnos en orden a la consecución de los objetivos previstos. Según sea el nivel de sus rendimientos califican a los escolares con palabras de aprobación o reprobación, lo cual supone

el restablecimiento de criterios para la valoración objetiva de los niveles de aprendizaje.

Se trata de poner en juego todo el potencial humano de los estudiantes para que lleguen a ser efectivas sus posibilidades reales. el éxito personal ya no depende tanto de las actitudes como del esfuerzo que se convierte así en el principal factor del aprendizaje o, dicho de otra manera, el aprendizaje es causa eficiente de la educación de las virtudes de los alumnos, tales como la fortaleza, la laboriosidad y la justicia.

5.4 Aplicaciones Generales

La capacidad de los navegadores Web para ejecutar applets de Java ha asegurado la continuidad del uso de Java por el gran público. Flash está más extendido para animaciones interactivas y los desarrolladores estén empezando a usar la tecnología AJAX también en este campo. Java suele usarse para aplicaciones más complejas como la zona de juegos de Yahoo, Yahoo! Games, o reproductores de video

En el PC de escritorio

Aunque cada vez la tecnología Java se acerca más y más al PC de sobremesa, las aplicaciones Java han sido relativamente raras para uso doméstico, por varias razones.[3]

- Las aplicaciones Java pueden necesitar gran cantidad de memoria física.
- La Interfaz Gráfica de Usuario (GUI) no sigue de forma estricta la *Guía para la Interfaz Humana'* (Human Interface Guidelines), así como tampoco aquella a la que estamos habitualmente acostumbrados. La apariencia de las fuentes no tiene las opciones de optimización activadas por defecto, lo que hace aparecer al texto como si fuera de baja calidad.
- Las herramientas con que cuenta el JDK no son suficientemente potentes para construir de forma simple aplicaciones potentes. Aunque el uso de herramientas como Eclipse, un IDE con licencia libre de alta calidad, facilita enormemente las tareas de desarrollo.
- Hay varias versiones del Entorno en Tiempo de Ejecución de Java, el JRE. Es necesario tener instalada la versión adecuada. El paquete JRE puede ser de tamaño considerable, 7Mbytes, lo que puede ser un inconveniente a la hora de descargarlo e instalarlo.
- Las aplicaciones basadas en la Web están tomando la delantera frente a aquellas que funcionan como entidades independientes. Las nuevas técnicas de programación producen aplicaciones basadas en un modelo en red cada vez más potentes.

Sin embargo hay aplicaciones Java cuyo uso está ampliamente extendido, como los NetBeans, el entorno de desarrollo (IDE) Eclipse, y otros programas como LimeWire y Azureus para intercambio de archivos. Java también es el motor que usa MATLAB para el renderizado de la interfaz gráfica y para parte del motor de cálculo. Las aplicaciones de escritorio basadas en la tecnología Swing y SWT (Standard Widget Toolkit) suponen una alternativa a la plataforma .Net de Microsoft.

La apariencia externa (el ‘”look and feel”’) de las aplicaciones GUI (Graphical User Interface) escritas en Java usando la plataforma Swing difiere a menudo de la que muestran aplicaciones nativas. Aunque el programador puede usar el juego de herramientas AWT (Abstract Windowing Toolkit) que genera objetos gráficos de la plataforma nativa, el AWT no es capaz de funciones gráficas avanzadas sin sacrificar la portabilidad entre plataformas; ya que cada una tiene un conjunto de APIs distinto, especialmente para objetos gráficos de alto nivel. Las herramientas de Swing, escritas completamente en Java, evitan este problema construyendo los objetos gráficos a partir de los mecanismos de dibujo básicos que deben estar disponibles en todas las plataformas. El inconveniente es el trabajo extra requerido para conseguir la misma apariencia de la plataforma destino. Aunque esto es posible (usando GTK+ y el Look-and-Feel de Windows), la mayoría de los usuarios no saben cómo cambiar la apariencia que se proporciona por defecto por aquella que se adapta a la de la plataforma. Mención merece la versión optimizada del Java Runtime que ha desarrollado Apple y que incluye en su sistema operativo, el Mac OS X. Por defecto implemente su propio look-and-feel (llamado Aqua), dando a las aplicaciones Swing ejecutadas en un Macintosh una apariencia similar a la que tendría si se hubiese escrito en código nativo.

5.5 Conclusiones

Podemos decir que en cuanto a las metodologías de enseñanza de la programación estamos aún poco documentados y preparados lo cual nos lleva a un campo difícil de entender e importante en el estudio de las ciencias de la computación ya que la programación es la parte inicial y fundamental que debe aprenderse con delicada sencillez para que no nos espante el mundo de los codigos.

CAPÍTULO 6

6. LENGUAJE PHP Y APLICACIONES WEB

6.1 Objetivo

Brindar los conocimientos para la creación de aplicaciones sobre Internet, utilizando el lenguaje de programación PHP.

6.2 Introducción

PHP (acrónimo de "PHP: Hypertext Preprocessor") es un lenguaje de "código abierto" interpretado, de alto nivel, embebido en páginas HTML y ejecutado en el servidor Web.

Lo mejor de usar PHP es que es extremadamente simple para el principiante, pero a su vez, ofrece muchas características avanzadas para los programadores profesionales. Aunque el desarrollo de PHP está concentrado en la programación de scripts en el lado del servidor, se puede utilizar para muchas otras cosas.

PHP puede ser utilizado en cualquiera de los principales sistemas operativos del mercado, Linux, muchas variantes de Unix (incluyendo HP-UX, Solaris y OpenBSD), Microsoft Windows, Mac OS X, RISC OS y etc. PHP soporta la mayoría de servidores Web de hoy en día, incluyendo Apache, Microsoft Internet Information Server, Personal Web Server, Netscape e iPlanet, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd y muchos otros. PHP tiene módulos disponibles para la mayoría de los servidores, para quellos otros que soporten el estándar CGI, PHP puede usarse como procesador CGI.

PHP tiene la posibilidad de usar programación procedimental o programación orientada a objetos. Aunque no todas las características estándar de la programación orientada a objetos están implementadas en la versión actual de PHP, muchas bibliotecas y aplicaciones están escritas íntegramente usando programación orientada a objetos.

PHP no se encuentra limitado a resultados en HTML. Entre las habilidades de PHP se incluyen: creación de imágenes, archivos PDF y películas Flash (usando libswf y Ming) sobre la marcha. También puede presentar otro resultados como XHTM y archivos XML. PHP puede auto generar éstos archivos y almacenarlos en el sistema de archivos en vez de presentarlos en la pantalla.

Quizás la característica más potente y destacable d-+e PHP es su soporte para una gran cantidad de bases de datos. Escribir un interfaz vía Web para una base de datos es una tarea simple con PHP. Las siguientes bases de datos están soportadas actualmente:

Adabas D, dBase, Empress, FilePro (solo lectura), Hyperwave, IBM DB2, Informix, Ingres, Internase, FrontBase, Msql, Direct MS-SQL, MySQL, ODBC, Oracle (OCI7 and OCI8), Ovrimos, PostgreSQL, Solid, Sybase, Veloces, Unix dbm.

PHP también cuenta con soporte para comunicarse con otros servicios usando protocolos tales como LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (en Windows) y muchos otros. También se pueden crear sockets puros. PHP soporta WDDX para el intercambio de datos entre lenguajes de programación en Web. Y hablando de interconexión, PHP puede utilizar objetos Java de forma transparente como objetos PHP Y la extensión de CORBA puede ser utilizada para acceder a objetos remotos.

PHP tiene unas características muy útiles para el procesamiento de texto, desde expresiones regulares POSIX extendidas o tipo Perl hasta procesadores de documentos XML. Para procesar y acceder a documentos XML, soportamos los estándares SAX y DOM. Puede utilizar la extensión XSLT para transformar documentos XML.

6.3 Desarrollo

Durante el desarrollo estudiaremos las principales caracteristicas del Lenguaje de Programación PHP.

6.3.1 Paginas WEB y HTML

WORLD WIDE WEB (WWW): Digamos, simplemente, que es un sistema de información, el sistema de información propio de Internet. Sus características son:

- Información por hipertexto: Diversos elementos (texto o imágenes) de la información que se nos muestra en la pantalla están vinculados con otras informaciones que pueden ser de otras fuentes. Para mostrar en pantalla esta otra información bastará con hacer clic sobre ellos.
- Gráfico: En la pantalla aparece simultáneamente texto, imágenes e incluso sonidos.
- Global: Se puede acceder a él desde cualquier tipo de plataforma, usando cualquier navegador y desde cualquier parte del mundo.
- Pública:Toda su información está distribuida en miles de ordenadores que ofrecen su espacio para almacenarla. Toda esta información es pública y toda puede ser obtenida por el usuario.
- Dinámica: La información, aunque esta almacenada, puede ser actualizada por el que la publico sin que el usuario deba actualizar su soporte técnico.
- Independiente: Dada la inmensa cantidad de fuentes, es independiente y libre.

NAVEGADOR: Es el programa que nos ofrece acceso a Internet. Debe ser capaz de comunicarse con un servidor y comprender el lenguaje de todas las herramientas que manejan la información de Web. Puede decirse que cada casa de software podría tener su navegador propio, aunque los mas populares sean Netscape e Internet Explorer.

SERVIDOR: Se encarga de proporcionar al navegador los documentos y medios que este solicita. Utiliza un protocolo HTTP para atender las solicitudes de archivos por parte de un navegador.

HTTP: Es el protocolo de transferencia de hipertexto, o sea, el protocolo que los servidores de World Wde Web utilizan para mandar documentos HTML a través de Internet.

URL: Es el Localizador Uniforme de Recursos, o dicho mas claramente, es la dirección que localiza una información dentro de Internet.

HTML: De momento, le basta saber que estas siglas se corresponden con la definición "Lenguaje para marcado de hipertexto". Más claro aún, se trata de un lenguaje para estructurar documentos a partir de texto en World Wide Web. Este lenguaje se basa en tags (instrucciones que le dicen al texto como deben mostrarse) y atributos (parámetros que dan valor al tag).

Organización de una Web

Para hacer una buena presentación Web lo ideal es crearnos un boceto inicial de la estructura. Si hacemos esto, no solo estamos procurando una presentación agradable y facilitando la tarea de navegar sino que también nos facilitamos el mantenimiento de futuras revisiones y modificaciones.

Objetivos

Lo primero que debemos hacer el fijarnos los objetivos que queremos alcanzar según la información que vayamos a aportar. Para crear nuestra primera página, estos objetivos deberían no ser muy pretenciosos o tener un sentido únicamente personal. Tener claros los objetivos nos ayudara a no plasmar contenidos confusos o innecesarios.

Contenidos

Una vez tenemos los objetivos, hay que organizar el contenido por temas o secciones, que se ajusten a nuestros objetivos, reuniendo las informaciones relacionadas bajo el mismo epígrafe. Es conveniente que los temas sean razonablemente cortos y si fuera necesario divida en subtemas. Si por el contrario tenemos temas muy cortos, lo correcto sería agruparlos bajo un encabezado de tema algo más general.

Primer paso

Una presentación Web consiste de una o más páginas Web que contienen texto y gráficos y que están vinculadas entre si creando un cuerpo de información. La página principal o página base es desde donde se comienza a visitar la presentación y su URL será la que figure como dirección de la presentación. Esta página base debe ofrecer un panorama general del contenido de la presentación.

Organización

Ha llegado la hora de estructurar la información recopilada en un conjunto de páginas Web. Podemos crearnos una estructura propia pero lo más lógico es guiarnos por una estructura clásica. Para más información sobre la estructura consulte las páginas "La estructura" y "Tipos de estructuras" de la guía de estilo.

Secuenciación

Consiste en decidir que contenido va en cada página, elaborar la trama de vínculos para navegar entre ellas e incluso, hacernos una idea de que tipo de gráficos vamos a poner y que ubicación van a tener. Para ello puede utilizarse un "Tablero de Secuencia", un esquema gráfico que nos ayudará a recordar en todo momento donde encaja cada página en el global de la presentación.
Revisión de objetivos

Finalmente y antes de ponernos a crear nuestra presentación Web, debemos prestar atención a que lo que tenemos plasmado en el "Tablero de Secuencia" cubre los objetivos que nos habíamos propuesto. Si es así, ya podemos comenzar a manejarnos con HTML.

El lenguaje HTML

Como ya se ha dicho, este lenguaje estructura documentos. La mayoría de los documentos tienen estructuras comunes (títulos, parrafos, listas...) que van a ser definidas por este lenguaje mediante tags. Cualquier cosa que no sea una tag es parte del documento mismo.
Este lenguaje no describe la apariencia del diseño de un documento sino que ofrece a cada plataforma que le de formato según su capacidad y la de su navegador (tamaño de la pantalla, fuentes que tiene instaladas...). Por ello y para no fustrarnos, no debemos diseñar los documentos basándonos en como lucen en nuestro navegador sino que debemos centrarnos en proporcionar un contenido claro y bien estructurado que resulte fácil de leer y entender.

No se desespere por lo que acaba de leer. HTML tiene dos ventajas que lo hacen prácticamente imprescindibles a la hora de diseñar una presentación web: Su compatibilidad y su facilidad de aprendizaje debido al reducido número de tags que usa.

Básicamente, los documentos escritos en HTML constan del texto mismo del documento y las tags que pueden llevar atributos. Esto llevado a la práctica, vendría a ser:

<tag> texto afectado </tag>

La tag del principio activa la orden y la última (que será la del principio precedida del signo /) la desactiva. No todas las tags tienen principio y final pero esto lo veremos más adelante.

EDITORES Y CONVERTIDORES

Antes de comenzar al trabajar sobre un editor, le recomendaría que visionase el código fuente de nuestra página principal. Todos los navegadores dan la opción de editarla (Menú ver / Código fuente). Si visita otras páginas y visualiza su código fuente encontrará similitudes en la forma en que están organizadas las páginas y en las tags utilizadas.

¿Dónde hay que editar el código fuente? Pues, si usted es usuario de Windows le bastaría con el Bloc de Notas y si utiliza Macintosh con el Simple Text. Si utiliza procesadores de texto más potentes debe guardar sus documentos como "solo texto" ya que HTML ignora todos los espacios en blanco. Una vez guardado convierta la extensión de texto por la extensión html o htm (en los sistemas DOS).

Los convertidores se utilizan para tomar los archivos de un procesador de textos y convertirlos a HTML. Pero debido a la propia limitación de este lenguaje, por muy elegante que hagamos un documento en nuestro procesador, un convertidor no obrará milagros y quizá acabe por crear cosas ilegibles en HTML. Además, la mayoría de los convertidores no convierten imágenes y no automatizan los vínculos hacia los documentos en Web debiendo corregir esto de manera manual.

A través de Internet o de revistas especializadas, usted podrá hacerse con editores y convertidores gratuitos o de muy reducidos costes. Quizá más adelante, cuando este acostumbrado a trabajar con HTML, puedan resultarle interesantes pero eso se lo dejo a su futura elección. De momento, hágame caso, si quiere aprender HTML use solo un procesador de texto simple.

DOCUMENTO HTML

<HTML> <HEAD> <TITLE> <BODY>

ESTRUCTURA BÁSICA DE UN DOCUMENTO HTML: Cabecera y cuerpo del documento

Tres son las tags que describen la estructura general de un documento y dan una información sencilla sobre él. Estas tags no afectan a la apariencia del documento y solo interpretan y filtran los archivos HTML.

1. <HTML>: Limitan el documento e indica que se encuentra escrito en este lenguaje.

2. <HEAD>: Especifica el prólogo del resto del archivo. Son pocas las tags que van dentro de ella, destacando la del titulo <TITLE> que será utilizado por los marcadores del navegador e identificará el contenido de la página. Solo puede haber un título por documento, preferiblemente corto aunque significativo, y no caben otras tags dentro de él. En head no hay que colocar nada del texto del documento.

3. <BODY>: Encierra el resto del documento, el contenido.

```
<HTML>
<HEAD>
<TITLE>Ejemplo 1</TITLE>
</HEAD>
<BODY>
Hola mundo
```

```
</BODY>
</HTML>
```

6.3.2 Primeros pasos

Saliendo de HTML

Para interpretar un archivo, PHP simplemente interpreta el texto del archivo hasta que encuentra uno de los caracteres especiales que delimitan el inicio de código PHP. El intérprete ejecuta entonces todo el código que encuentra, hasta que encuentra una etiqueta de fin de código, que le dice al intérprete que siga ignorando el código siguiente. Este mecanismo permite embeber código PHP dentro de HTML: todo lo que está fuera de las etiquetas PHP se deja tal como está, mientras que el resto se interpreta como código.

Hay cuatro conjuntos de etiquetas que pueden ser usadas para denotar bloques de código PHP. De estas cuatro, sólo 2 (<?php. . .?> y <script language="php">. . .</script>) están siempre disponibles; el resto pueden ser configuradas en el fichero de php.ini para ser o no aceptadas por el intérprete. Mientras que el formato corto de etiquetas (short-form tags) y el estilo ASP (ASP-style tags) pueden ser convenientes, no son portables como la versión de formato largo de etiquetas. Además, si se pretende embeber código PHP en XML o XHTML, será obligatorio el uso del formato <?php. . .?> para la compatibilidad con XML.

Las etiquetas soportadas por PHP son:

1. <?php echo("si quieres servir documentos XHTML o XML, haz como aquí\n"); ?>

2. <? echo ("esta es la más simple, una instrucción de procesado SGML \n"); ?>
   <?= expression ?> Esto es una abreviatura de "<? echo expression ?>"

3. <script language="php">
     echo ("muchos editores (como FrontPage) no
          aceptan instrucciones de procesado");
   </script>

4. <% echo ("Opcionalmente, puedes usar las etiquetas ASP"); %>
   <%= $variable; # Esto es una abreviatura de "<% echo . . ." %>

PHP soporta el estilo de comentarios de 'C', 'C++' y de la interfaz de comandos de Unix.

Por ejemplo:

```
<?php
   echo "This is a test"; // This is a one-line c++ style comment
   /* This is a multi line comment
      yet another line of comment */
   echo "This is yet another test";
   echo "One Final Test"; # This is shell-style style comment
?>
```

Los estilos de comentarios de una línea actualmente sólo comentan hasta el final de la línea o el bloque actual de código PHP, lo primero que ocurra.

```
<h1>This is an <?php # echo "simple";?> example.</h1>
<p>The header above will say 'This is an example'.
```

Hay que tener cuidado con no anidar comentarios de estilo 'C', algo que puede ocurrir al comentar bloques largos de código.

```
<?php
 /*
   echo "This is a test"; /* This comment will cause a problem */
 */
?>
```

Los estilos de comentarios de una línea actualmente sólo comentan hasta el final de la línea o del bloque actual de código PHP, lo primero que ocurra. Esto implica que el código HTML tras // ?> será impreso: ?> sale del modo PHP, retornando al modo HTML, el comentario // no le influye.

VARIABLES.

En PHP las variables se representan como un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable valido tiene que empezar con una letra o una raya (underscore), seguido de cualquier número de letras, números y rayas. Como expresión regular se podría expresar como: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

ÁMBITO DE LAS VARIABLES.

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los archivos incluidos y los requeridos.

Por ejemplo:

```
<?php
$a = 1;
include "b.inc";
?>
```

Aquí, la variable $a dentro del script incluido b.inc. De todas formas, dentro de las funciones definidas por el usuario aparece un ámbito local a la función. Cualquier variable que se use dentro de una función está, por defecto, limitada al ámbito local de la función.

Por ejemplo:

```
<?php
$a = 1; /* global scope */

function Test()
{
   echo $a; /* reference to local scope variable */
}

Test();
?>
```

Este script no producirá salida, ya que la orden echo utiliza una versión local de la variable $a, a la que no se ha asignado ningún valor en su ámbito. Puede que usted note que hay una pequeña diferencia con el lenguaje C, en el que las variables globales están disponibles automáticamente dentro de la función a menos que sean expresamente sobre escritas por una definición local. Esto puede causar algunos problemas, ya que la gente puede cambiar variables globales inadvertidamente. En PHP, las variables globales deben ser declaradas globales dentro de la función si van a ser utilizadas dentro de dicha función.

Por Ejemplo:

```
<?php
$a = 1;
$b = 2;

function Sum()
{
   global $a, $b;
```

```
  $b = $a + $b;
}

Sum();
echo $b;
?>
```

El script anterior producirá la salida "3". Al declarar $a y $b globales dentro de la función, todas las referencias a tales variables se referirán a la versión global. No hay límite al número de variables globales que se pueden manipular dentro de una función.
Un segundo método para acceder a las variables desde un ámbito global es usando la matriz $GLOBALS. El ejemplo anterior se puede rescribir así:

Por Ejemplo:

```
<?php
$a = 1;
$b = 2;

function Sum()
{
  $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum();
echo $b;
?>
```

La matriz $GLOBALS es una matriz asociativa con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento de la matriz. $GLOBALS existe en cualquier ámbito, esto pasa porque $GLOBALS es una superglobal. Aqui teneis un ejemplo que demuestra el poder de las superglobales:

```
<?php
function test_global()
{
  // Most predefined variables aren't "super" and require
  // 'global' to be available to the functions local scope.
  global $HTTP_POST_VARS;

  print $HTTP_POST_VARS['name'];

  // Superglobals are available in any scope and do
```

```
  // not require 'global'.  Superglobals are available
  // as of PHP 4.1.0
  print $_POST['name'];
}
?>
```

Otra característica importante del ámbito de las variables es la variable static. Una variable estática existe sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito. Consideremos el siguiente ejemplo:

```
<?php
function Test ()
{
  $a = 0;
  echo $a;
  $a++;
}
?>
```

Esta función tiene poca utilidad ya que cada vez que es llamada asigna a $a el valor 0 y representa un "0". La sentencia $a++, que incrementa la variable, no sirve para nada, ya que en cuanto la función termina la variable $a desaparece. Para hacer una función útil para contar, que no pierda la pista del valor actual del conteo, la variable $a debe declararse como estática:

```
<?php
function Test()
{
  static $a = 0;
  echo $a;
  $a++;
}
?>
```

Ahora, cada vez que se llame a la función Test(), se representará el valor de $a y se incrementará.

Las variables estáticas también proporcionan una forma de manejar funciones recursivas. Una función recursiva es la que se llama a sí misma. Se debe tener cuidado al escribir una función recursiva, ya que puede ocurrir que se llame a sí misma indefinidamente. Hay que asegurarse de implementar una forma adecuada de terminar la recursión. La siguiente función cuenta recursivamente hasta 10, usando la variable estática $count para saber cuándo parar:

```
<?php
function Test()
{
  static $count = 0;

  $count++;
  echo $count;
  if ($count < 10) {
     Test ();
  }
  $count--;
}
?>
```

6.3.3 Operadores y Sentencias.

Tipos de Operadores

- Operadores de Aritmética
- Operadores de Asignación
- Operadores Bit a Bit
- Operadores de Comparación
- Operadores de Control de Errores
- Operadores de ejecución
- Operadores de Incremento/Decremento
- Operadores de Lógica
- Operadores de Cadena
- Operadores de Matrices
- Operadores de Tipo

Un operador es algo a lo que usted entrega uno o más valores (o expresiones, en jerga de programación) y produce otro valor (de modo que la construcción misma se convierte en una expresión). Así que puede pensar sobre las funciones o construcciones que devuelven un valor (como print) como operadores, y en aquellas que no devuelven nada (como echo) como cualquier otra cosa.

Existen tres tipos de operadores. En primer lugar se encuentra el operador unario, el cual opera sobre un único valor, por ejemplo ! (el operador de negación) o ++ (el operador de incremento). El segundo grupo se conoce como operadores binarios; éste grupo contiene la mayoría de operadores que soporta PHP.

El tercer grupo consiste del operador ternario: ?:. Éste debe ser usado para seleccionar entre dos expresiones, en base a una tercera, en lugar de seleccionar dos sentencias o rutas de ejecución. Rodear las expresiones ternarias con paréntesis es una muy buena idea.

PRECEDENCIA DE OPERADORES

La precedencia de un operador indica qué tan "cerca" se agrupan dos expresiones. Por ejemplo, en la expresión 1 + 5 * 3, la respuesta es 16 y no 18, ya que el operador de multiplicación ("*") tiene una mayor precedencia que el operador de adición ("+"). Los paréntesis pueden ser usados para marcar la precedencia, si resulta necesario. Por ejemplo: (1 + 5) * 3 evalúa a 18. Si la precedencia de los operadores es la misma, se utiliza una asociación de izquierda a derecha.
La siguiente tabla lista la precedencia de los operadores, con aquellos de mayor precedencia listados al comienzo de la tabla. Los operadores en la misma línea tienen la misma precedencia, en cuyo caso su asociatividad decide el orden para evaluarlos.

Precedencia de Operadores

Asociatividad	Operadores	Información Adicional
no-asociativo	new	new
derecha	[	array()
no-asociativos	++ --	incremento/decremento
no-asociativos	! ~ - (int) (float) (string) (array) (object) @	tipos
izquierda	* / %	aritmética
izquierda	+ - .	aritmética, y cadena
izquierda	<< >>	manejo de bits
no-asociativos	< <= > >=	comparación
no-asociativos	== != === !==	comparación
izquierda	&	manejo de bits, y referencias
izquierda	^	manejo de bits
izquierda	|	manejo de bits
izquierda	&&	lógicos
izquierda	||	lógicos
izquierda	? :	ternario
derecha	= += -= *= /= .= %= &= |= ^= <<= >>=	asignación
izquierda	and	lógicos
izquierda	xor	lógicos

Asociatividad	Operadores	Información Adicional
izquierda	or	lógicos
izquierda	,	varios usos

Tabla 6.1 Precedencia de operadores en PHP

La asociatividad de izquierda quiere decir que la expresión es evaluada desde la izquierda a la derecha, la asociatividad de derecha quiere decir lo contrario.

6.3.4 Funciones y bibliotecas

Funciones definidas por el usuario

Una función se puede definir con la siguiente sintaxis:

Ejemplo.

```
<?php
function foo ($arg_1, $arg_2, ..., $arg_n)
{
   echo "Funci&oacute;n de ejemplo.\n";
   return $retval;
}
?>
```

Cualquier instrucción válida de PHP puede aparecer en el cuerpo de la función, incluso otras funciones y definiciones de clases.

PARÁMETROS DE LAS FUNCIONES

La información puede suministrarse a las funciones mediante la lista de parámetros, una lista de variables y/o constantes separadas por comas.

PARÁMETROS POR VALOR.

PHP soporta pasar parámetros por valor (el comportamiento por defecto), por referencia, y parámetros por defecto. Listas de longitud variable de parámetros sólo están soportadas en PHP4 y posteriores; ver Listas de longitud variable de parámetros y la referencia de las funciones func_num_args(), func_get_arg(), y

func_get_args() para más información. Un efecto similar puede conseguirse en PHP3 pasando un array de parámetros a la función.

Ejemplo. Pasando matrices a funciones:

```
<?php
function takes_array($input)
{
   echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
?>
```

PASAR PARÁMETROS POR REFERENCIA

Por defecto, los parámetros de una función se pasan por valor (de manera que si cambias el valor del argumento dentro de la función, no se ve modificado fuera de ella). Si deseas permitir a una función modificar sus parámetros, debes pasarlos por referencia.
Si quieres que un parámetro de una función siempre se pase por referencia, puedes anteponer un ampersand (&) al nombre del parámetro en la definición de la función:

Ejemplo. Pasando parámetros de funciones por referencia:

```
<?php
function add_some_extra(&$string)
{
   $string .= ' y algo m&aacute;s.';
}
$str = 'Esto es una cadena, ';
add_some_extra($str);
echo $str;    // Saca 'Esto es una cadena, y algo m&aacute;s.'
?>
```

PARÁMETROS POR DEFECTO.

Una función puede definir parámetros por defecto para valores escalares.

Ejemplo. Parámetros por defecto.

```
<?php
function makecoffee ($type = "cappucino")
{
   return "Hacer una taza de $type.\n";
}
echo makecoffee ();
```

```
echo makecoffee ("espresso");
?>
```

DEVOLVIENDO VALORES

Los valores se retornan usando la instrucción opcional return. Puede devolverse cualquier tipo de valor, incluyendo listas y objetos.

Ejemplo. Uso de return.

```
<?php
function square ($num)
{
   return $num * $num;
}
echo square (4);  // saca '16'.
?>
```

FUNCIONES VARIABLES.

PHP soporta el concepto de funciones variable, esto significa que si una variable tiene unos paréntesis añadidos al final, PHP buscará una función con el mismo nombre que la evaluación de la variable, e intentará ejecutarla. Entre otras cosas, esto te permite implementar retrollamadas (callbacks), tablas de funciones y demás.
Las funciones varibles no funcionarán con construcciones del lenguaje, tal como echo(), print(), unset(), isset(), empty(), include(), require() y derivados. Se necesitará usar una función propia para utilizar cualquiera de estos constructores como funciones variables.

Ejemplo de funciones variables:

```
<?php
function foo()
{
   echo "In foo()<br>\n";
}

function bar($arg = '')
{
   echo "In bar(); argument was '$arg'.<br>\n";
}
```

```
// This is a wrapper function around echo
function echoit($string)
{
   echo $string;
}

$func = 'foo';
$func();        // This calls foo()

$func = 'bar';
$func('test');  // This calls bar()

$func = 'echoit';
$func('test');  // This calls echoit()
?>
```

También se puede llamar a un método de un objeto usando la característica variable de las funciones.

FUNCIONES INTERNAS (INCORPORADAS)

PHP tiene incorporadas muchas funciones y construcciones. Existen también funciones que requieren extensiones específicas de PHP para que no fallen con un error fatal del tipo "undefined function". Por ejemplo, para usar funciones image, tal como imagecreatetruecolor(), se necesita compilar PHP con soporte para GD. O para usar mysql_connect() se necesita compilar PHP con soporte para MySQL. Existen muchas funciones en el núcleo de PHP que se incluyen en cada versión de PHP, como las funciones string y variable. Una llamada a la función phpinfo() ó get_loaded_extensions() mostrará que extensiones están cargadas en tu versión de PHP. Tener también en cuenta que muchas extensiones se encuentran activadas por defecto y que el manual de PHP se encuentra dividido en partes, según estas extensiones.

6.3.5 Procesado de formularios

Existen tres posibles configuraciones para el proceso de formularios:

Un fichero HMTL para enviar la información del formulario y un archivo PHP para recibirla y procesarla.

- Se crea un archivo HTML con el diseño del formulario.

  ```
  <form action = "procesar.php" method="post">
        Nombre: <input type"text" name="nombre" br>
        <input type = "submit">
  </form>
  ```

  Se debe de declarar en la etiqueta form action = "archivo.php", lo cual nos indique a que página vamos a enviar la información para procesar y en cada input type declaramos name = "variable" lo cual nos indica el nombre de la variable en que va a ser almacenado cada dato.

  Después se creará en archivo PHP (archivo.php) que procesará los datos.

  Ejemplo.

  ```
  <?php
        echo $nombre;
  ?>
  ```

- Un fichero php para enviar la información y un fichero php para recibirla.

  Se usa un archivo PHP para enviar la información:

  ```
  <form action = "procesar.php" method="post">
        Nombre: <input type"text" name="nombre" value=<?= $nombre?>>
        <br><input type = "submit">
   </form>
  ```

  Se debe de declarar en la etiqueta form action = "archivo.php", lo cual nos indique a que página vamos a enviar la información para procesar y en cada input type declaramos name = "variable" lo cual nos indica el nombre de la variable en que va a ser almacenado cada dato, con el valor por default que asigna la variable de PHP $nombre.

  Después se creará en archivo PHP (archivo.php) que procesará los datos.

Ejemplo.

```
<?php
        echo $nombre;
?>
```

- Un archivo PHP para enviar la información y el mismo archivo PHP para recibirla y procesarla.

  Realizamos un solo fichero PHP que va a enviar y recibir los datos.

  Ejemplo.

```
<?php
if($enviar!="Enviar")
            {
?>
<form action="<?=$SERVER[PHP_SELF]" method="POST">
Nombre: <input type="text" name="nombre"><br>
Codigo: <input type="text" name="codigo"> 
<select name="carrera">
            <option>Ingenieria de sistemas
            <option>Ingenieria electronica
</select>
<br><br>
<input type="submit" name="enviar" value="Enviar">
 
<input type="reset" name="reset">
</form>
<?php
            }
else    {
echo "Nombre: $nombre<br>";
echo "Codigo: $codigo<br>";
echo "Carrera: $carrera<br>";
            }
?>
```

En este caso en la etiqueta form action se declara el mismo nombre del archivo para que los datos sean procesados por este mismo y nos apoyamos en la variable dada al tipo submit para realizar un escape avanzado del código, en caso de que el usuario no haya pulsado el botón de submit como cuando la página ha sido cargada por primera vez, nos salimos del script y generamos el formulario o en caso contrario volvemos al script y procesamos los datos.

6.3.6 Acceso a Bases de Datos

Para la realización de este curso sobre PHP con acceso a base de datos hemos elegido la base de datos MySQL por ser gratuita y por ser también la mas empleada en entornos UNIX, para lo cual el servidor donde tenemos alojadas las páginas nos tiene que proporcionar herramientas para crearla o acceso al Telnet para que la creemos por nosotros mismos.

El comando para crear una base de datos MySQL es el siguiente:

mysqladmin -u root create base_datos

Con este comando conseguimos crear la una base de datos en el servidor de bases de datos de nuestro servidor.
Una vez conseguido esto debemos crear las tablas en la base de datos, la descripción de las tablas contienen la estructura de la información que almacenaremos en ellas. Para lo cual usaremos en lenguaje de consultas SQL común para todas las bases de datos relacionales.

En este ejemplo creamos una tabla llamada prueba con 3 campos: un campo identificador, que nos servirá para identificar unívocamente una fila con el valor de dicho campo, otro campo con el nombre de una persona y por último un campo con el apellido de la persona.

Para crear la tabla puede usar la herramienta de administración de MySQL de su servidor web o puede escribir un fichero de texto con el contenido de la sentencia SQL equivalente y luego decirle al motor de base de datos que la ejecute con la siguiente instrucción:

mysql -u root base_datos <prueba.sql

prueba.sql

```
CREATE TABLE prueba (
ID_Prueba int(11) DEFAULT '0' NOT NULL auto_increment,
Nombre varchar(100),
Apellidos varchar(100),
PRIMARY KEY (ID_Prueba),
UNIQUE ID_Prueba (ID_Prueba)
);
```

Una vez que tenemos creada la base de datos en nuestro servidor, el siguiente paso es conectarnos a la misma desde una página PHP. Para ello PHP nos proporciona una serie de instrucciones para acceder a bases de datos MySQL.

```
<!-- Manual de PHP de WebEstilo.com -->
<html>
<head>
  <title>Ejemplo de PHP</title>
</head>
<body>
<?php
function Conectarse()
{
  if (!($link=mysql_connect("localhost","usuario","Password")))
  {
    echo "Error conectando a la base de datos.";
    exit();
  }
  if (!mysql_select_db("base_datos",$link))
  {
    echo "Error seleccionando la base de datos.";
    exit();
  }
  return $link;
}

$link=Conectarse();
echo "Conexión con la base de datos conseguida.<br>";

mysql_close($link); //cierra la conexion
?>
</body>
</html>
```

Ver código fuente

Ejecutar ejemplo

Al ejecutar la instrucción mysql_connect creamos un vínculo entre la base de datos y la pagina PHP, este vínculo será usado posteriormente en las consultas que hagamos a la base de datos.

Finalmente, una vez que hemos terminado de usar el vínculo con la base de datos, lo liberaremos con la instrucción mysql_close para que la conexión no quede ocupada.

Una vez que nos hemos conectado con el servidor de bases de datos, ya podemos realizar consultas a las tablas de la base de datos.

Para facilitar la programación hemos separado la función de conexión en una librería a parte, de tal manera que la incluiremos en todas las páginas que accedan a la base de datos.

conex.phtml

```
<!-- Manual de PHP de WebEstilo.com -->
<?php
function Conectarse()
{
  if (!($link=mysql_connect("localhost","usuario","Password")))
  {
    echo "Error conectando a la base de datos.";
    exit();
  }
  if (!mysql_select_db("base_datos",$link))
  {
    echo "Error seleccionando la base de datos.";
    exit();
  }
  return $link;
}
?>
```

Ver código fuente

```
<!-- Manual de PHP de WebEstilo.com -->
<html>
<head>
  <title>Ejemplo de PHP</title>
</head>
<body>
<H1>Ejemplo de uso de bases de datos con PHP y MySQL</H1>
<?php
  include("conex.phtml");
  $link=Conectarse();
  $result=mysql_query("select * from prueba",$link);
?>
  <TABLE BORDER=1 CELLSPACING=1 CELLPADDING=1>
    <TR><TD> Nombre</TD><TD> Apellidos </TD></TR>
<?php

  while($row = mysql_fetch_array($result)) {
    printf("<tr><td> %s</td><td> %s </td></tr>",
$row["Nombre"],$row["Apellidos"]);
  }
  mysql_free_result($result);
  mysql_close($link);
?>
</table>
</body>
</html>
```

Ejecutar ejemplo Ver código fuente

En este ejemplo hemos utilizado 3 instrucciones nuevas: mysql_query, mysql_fetch_array y mysql_free_result. Con la instrucción mysql_query hemos hecho una consulta a la base de datos en el lenguaje de consultas SQL, con la instrucción mysql_fetch_array extraemos los datos de la consulta a un array y con mysql_free_result liberamos la memoria usada en la consulta.

Hasta ahora nos hemos conectado a una base de datos y hemos hecho consultas a la misma, ahora presentaremos como introducir nuevo registros en la base de datos.

Para ello usaremos un formulario y en el ACTION del FORM <FORM ACTION="programaPHP"> indicaremos que debe ser procesado una pagina PHP, esta página lo que hará será introducir los datos del formulario en la base de datos.
ejem07d.phtml

```
<!-- Manual de PHP de WebEstilo.com -->
<html>
<head>
  <title>Ejemplo de PHP</title>
</head>
<body>
<H1>Ejemplo de uso de bases de datos con PHP y MySQL</H1>
<FORM ACTION="procesar.phtml">
<TABLE>
<TR>
  <TD>Nombre:</TD>
  <TD><INPUT TYPE="text" NAME="nombre" SIZE="20"
MAXLENGTH="30"></TD>
</TR>
<TR>
  <TD>Apellidos:</TD>
  <TD><INPUT TYPE="text" NAME="apellidos" SIZE="20"
MAXLENGTH="30"></TD>
</TR>
</TABLE>
<INPUT TYPE="submit" NAME="accion" VALUE="Grabar">
</FORM>
<hr>
<?php
  include("conex.phtml");
  $link=Conectarse();
  $result=mysql_query("select * from prueba",$link);
?>
  <TABLE BORDER=1 CELLSPACING=1 CELLPADDING=1>
    <TR><TD> <B>Nombre</B></TD>
<TD> <B>Apellidos</B> </TD></TR>
<?php

  while($row = mysql_fetch_array($result)) {
    printf("<tr><td> %s</td> <td> %s </td></tr>",
$row["Nombre"], $row["Apellidos"]);
  }
  mysql_free_result($result);
  mysql_close($link);
?>
</table>
</body>
</html>
```

Ejecutar ejemplo Ver código fuente

procesar.phtml

```
<?php
  include("conex.phtml");
  $link=Conectarse();
  $nombre=$_GET['nombre'];
  $apellidos=$_GET['apellidos'];
  mysql_query("insert into prueba (Nombre,Apellidos) values
('$nombre','$apellidos')",$link);

  header("Location: ejem07d.phtml");
?>
```

La primera página PHP ejem07d.phtml es un formulario que nos permite introducir nombre y apellido para añadirlo a la base de datos, seguido de una consulta que nos muestra el contenido de la tabla prueba. El formulario llama a la pagina procesar.phtml que añadirá los datos a la tabla.

La segunda página procesar.phtml se conecta a la base de datos y añade un nuevo registro con la instrucción insert del lenguaje de base de datos SQL. Una vez el registro se ha añadido se vuelve a cargar la página ejem07d.phtml
finalmente, para cerrar el ciclo, nos queda el borrado de registros. El borrado de registros el uno de los procesos más sencillos.
Para indicar que elemento vamos a borrar hemos usado un enlace a la página borra.phtml pasándole el ID_Prueba de cada registro, de esta manera la página borra.phtml sabe que elemento de la tabla ha de borrar.

ejem07e.phtml

```
<!-- Manual de PHP de WebEstilo.com -->
<html>
<head>
  <title>Ejemplo de PHP</title>
</head>
<body>
<H1>Ejemplo de uso de bases de datos con PHP y MySQL</H1>

<?php
  include("conex.phtml");
  $link=Conectarse();
  $result=mysql_query("select * from prueba",$link);
?>
  <TABLE BORDER=1 CELLSPACING=1 CELLPADDING=1>
    <TR><TD> <B>Nombre</B></TD>
<TD> <B>Apellidos</B> </TD>
<TD> <B>Borrar</B> </TD></TR>
<?php
```

```
  while($row = mysql_fetch_array($result)) {
     printf("<tr><td> %s</td><td> %s </td><td><a
href=\"borra.phtml?id=%d\">Borra</a></td></tr>",
$row["Nombre"],$row["Apellidos"],$row["ID_Prueba"]);
  }
  mysql_free_result($result);
  mysql_close($link);
?>
</table>
</body>
</html>
```

Ejecutar ejemplo Ver código fuente

borra.phtml

```
<?php
  include("conex.phtml");
  $link=Conectarse();
  $id=$_GET['id'];
  mysql_query("delete from prueba where ID_Prueba = $id",$link);

  header("Location: ejem07e.phtml");
?>
```

La página borra.phtml se conecta a la base de datos y borra el registro indicado en la variable $id que ha sido pasado desde la página ejem07e.phtml. Una vez el registro se ha borrado se vuelve a cargar la página ejem07e.phtml

6.3.7 Manejo de Sesiones

El apoyo que PHP proporciona para las sesiones consiste en una forma de conservar ciertos datos a lo largo de los subsiguientes accesos, lo cual le permite construir aplicaciones más personalizadas e incrementar el atractivo de su sitio Web.

A cada visitante que accede al sitio se le asigna un identificador único, llamado "session id" (identificador de sesión). Éste se almacena en una cookie por parte del usuario o se propaga en la URL.

El soporte de las sesiones le permite registrar un número arbitrario de variables que se conservarán en las siguientes peticiones. Cuando un visitante acceda al sitio Web, PHP comprobará automáticamente (si session.auto_start está puesto a 1) o cuando usted lo especifique (de forma explícita mendiante session_start() o implícita a través de session_register()) si se le ha enviado un "session id" específico con su petición, en cuyo caso se recrean las variables que se habían guardado anteriormente.

Todas las variables registradas son almacenadas tras finalizar la petición. Las variables que están indefinidas se marcan como no definidas. En los subsiguientes accesos, no estarán definidas por el módulo de sesiones a menos que el usuario las defina más tarde.

Las opciones de configuración track_vars y register_globals influyen notablemente en la forma en que las variables de la sesión se almacenan y restauran.

Si track_vars está activado y register_globals está desactivado, sólo los miembros del vector asociativo global $HTTP_SESSION_VARS pueden ser registrados como variables de la sesión. Las variables restauradas de la sesión sólo estarán disponibles en el vector $HTTP_SESSION_VARS.

Se recomienda usar $_SESSION (o $HTTP_SESSION_VARS con PHP 4.0.6 o inferior) por seguridad y para hacer el código más legible. Con $_SESSION o $HTTP_SESSION_VARS, no es necesario usar las funciones session_register() / session_unregister() / session_is_registered(). Los usuarios pueden acceder a una variable de la sesión como si se tratase de una variable normal.

Si register_globals está activado, todas las variables globales pueden ser registradas como variables de la sesión, y las variables de la sesión serán restauradas a sus correspondientes variables globales. Como PHP debe saber qué variables globales están registradas como variables de la sesión, los usuarios deben registrar las variables con la función session_register(), mientras que con $HTTP_SESSION_VARS/$_SESSION no es necesario usar session_register().

Si track_vars y register_globals están activados, las variables globales y las entradas de $HTTP_SESSION_VARS/$_SESSION harán referencia al mismo valor para variables ya registradas.

Si el usuario utiliza session_register() para registrar una variable, el vector $HTTP_SESSION_VARS/$_SESSION no contendrá esa variable hasta que se cargue de los datos de la sesión. (hasta la próxima petición).

Hay dos formas de propagar un "session id". Cookies y como parámetro en la URL.

El módulo de sesiones admite ambas formas. Las Cookies son la mejor opción, pero como no son fiables (los clientes no están obligados a aceptarlas), no podemos confiar en ellas. El segundo método incrusta el "session id" directamente en la URL.

Algunas de las funciones predefinidas en PHP para el manejo de Sesiones son:

session_cache_expire -- Devuelve la caducidad actual del caché
session_cache_limiter -- Lee y/o cambia el limitador del caché actual
session_commit -- Alias of session_write_close()
session_decode -- Decodifica los datos de una sesión a partir de una cadena
session_destroy -- Destruye todos los datos guardados en una sesión
session_encode -- Codifica los datos de la sesión actual en una cadena
session_get_cookie_params -- Obtiene los parámetros de la cookie de la sesión
session_id -- Lee y/o cambia el session id actual
session_is_registered -- Comprueba si una variable está registrada en la sesión
session_module_name -- Lee y/o cambia el módulo de la sesión actual
session_name -- Lee y/o cambia el nombre de la sesión actual
session_regenerate_id -- Actualizar el id de sesión actual con una recién generada
session_register -- Registrar una o más variables globales con la sesión actual

session_save_path -- Lee y/o cambia la ruta donde se guardan los datos de la sesión actual
session_set_cookie_params -- Cambia los parámetros de la cookie de la sesión
session_set_save_handler -- Establece unas funciones para el almacenamiento de los datos de la sesión a nivel de usuario
session_start -- Inicializar los datos de una sesión

session_unregister -- Desregistrar una variable de la sesión actual
session_unset -- Elimina todas las variables de la sesión
session_write_close -- Escribe los datos de la sesión y la finaliza

6.4 Aplicaciones Generales

PHP puede hacer cualquier cosa que se pueda hacer con un script CGI, como procesar la información de formularios, generar páginas con contenidos dinámicos, o enviar y recibir cookies.

Existen tres campos en los que se usan scripts escritos en PHP:

- Scripts del lado del servidor. Este es el campo más tradicional y el principal foco de trabajo. Se necesitan tres cosas para que esto funcione. El intérprete PHP (CGI ó módulo), un servidor Web (Apache) y un navegador. Es necesario correr el servidor Web con PHP instalado. El resultado del programa PHP se puede obtener a través del navegador, conectándose con el servidor Web.
- Scripts en la línea de comandos. Puede crear un script PHP y correrlo sin ningún servidor Web o navegador. Solamente necesita el intérprete PHP para usarlo de esta manera. Este tipo de uso es ideal para scripts ejecutados regularmente desde cron (en *nix o Linux) o el Planificador de tareas (en Windows). Estos scripts también pueden ser usados para tareas simples de procesamiento de texto.
- Escribir aplicaciones de interfaz gráfica. Probablemente PHP no sea el lenguaje más apropiado para escribir aplicaciones gráficas, pero si se conoce bien PHP, y se quisiera utilizar algunas características avanzadas en programas clientes, puede utilizar PHP-GTK para escribir dichos programas. También es posible escribir aplicaciones independientes de una plataforma.

6.5 Conclusiones

Las posibilidades del lenguaje PHP son excelentes, hasta el punto que es posible crear en PHP todas las aplicaciones que se podrían crear con unos script CGI. La diferencia principal entre los dos es que el primero hace mucho más simple la conexión y las preguntas con las bases de datos; el PHP3 soporta las siguientes bases de datos:

1. Adabas D
2. InterBase
3. Solid
4. dBase
5. mSQL
6. Sybase
7. Empress
8. MySQL
9. Velocis
10. FilePro
11. Oracle
12. Unix dbm
13. Informix
14. PostgreSQL

Al igual que con los CGI, con el PHP es posible utilizar los protocolos de red más famosos como IMAP, SMTP, POP3 e incluso HTTP, o utilizar los socket (enchufes).

Es por todo esto que el Lenguaje PHP es actualmente utilizado principalmente en el desarrollo de páginas Web.

CONCLUSIONES GENERALES

El uso de los diferentes lenguajes de programación para la enseñanza del desarrollo de software es cada vez de mayor relevancia, debido al bajo costo que automatizar un proceso implica.

Es por ello la relevancia e importancia que tiene la actualización de los profesores de nivel bachillerato de la unam, esto en cuanto a lenguajes de programación, sin olvidar los metodos y tecnicas que existen para poder transmitir los conocimientos en cuanto a programación se refiere.

Por otro lado las aplicaciones para Web creadas con PHP como lenguaje de programación y algún manejador de base de datos como son Mysql o PostgreSQL han llegado a ser de una popularidad trascendente en el entorno de desarrollos Web, no sólo por su fácil implementación y rápida curva de aprendizaje sino por que son herramientas fiables y seguras, por estas caracteristicas es de fundamental estudio este tema en el diplomado.

Los conocimientos adquiridos mediante los módulos que el diplomado comprende son suficientes para comenzar a introducir a los alumnos en el desarrollo e implementación de sistemas y así aprovechar las ventajas que tienen las herramientas, manejadores de bases de datos, y lenguajes de programación.

BIBLIOGRAFÍA

BALTHASAR, N Abstracción ontological y metafísica: 1935.

Charte Francisco. **“Delphi 5”.**
1ª. Edición, año 1999. Madrid.
Ed. Anaya Multimedia, 352 p

Decaer, Rick. **“Programación con Java”.**
2ª. Edición, año 2001. México.
Ed. Hamilton Collage, 606 p

Peñaloza Romero Ernesto. **“Didáctica de la Programación”.**
1ª. Edición, año 2003. México.

Pérez Medel Marcelo. **“Lenguaje de Programación Pascal”.**
1ª. Edición, año 2003. México.

Ramírez Montalvo Ernesto. **“Solución de Problemas y algoritmos”.**
1ª. Edición, año 2003. México.

Sanchi Llorca Francisco J. **“Programación con el lenguaje pascal”.**
2ª. Edición, año 1991. Madrid.
Ed. Paraninfo, 362 p