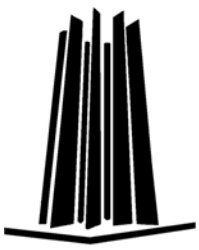
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

"ACTUALIZACIÓN PARA DOCENTES EN CÓMPUTO DEL NIVEL BACHILLERATO DE LA UNAM ORIENTADO A LA PROGRAMACIÓN ABARCANDO LOS ASPECTOS DIDÁCTICOS"

T R A B A J O E S C R I T O
EN LA MODALIDAD DE SEMINARIOS Y CURSOS DE ACTUALIZACIÓN Y CAPACITACIÓN PROFESIONAL
QUE PARA OBTENER EL TÍTULO DE:
I N G E N I E R O E N C O M P U T A C I Ó N
P R E S E N T A :
K A R L A O R T E G A V E L A S C O

ASESOR: M. EN C. MARCELO PÉREZ MEDEL.



MÉXICO, 2006.

AGRADECIMIENTOS

Doy infinitas gracias a Dios por darme la serenidad y el valor de concluir un sueño y darme las facultades para pensar en el futuro.

Gracias a mi padre por su amor, su apoyo y sobre todo por su gran ejemplo como profesional.

Gracias a mi madre por estar conmigo fielmente en espíritu, en todo mis momentos de alegrías y derrotas.

A mis hermanos Armando y Victor, así como a sus familias, quienes durante todos estos años confiaron en mi comprendiendo mis ideales.

A Enrique que sin él no lo hubiera podido lograr, gracias por su comprensión en cada momento de desaliento, por su ayuda que sin ella no estaría aquí y sobre todo por su amor y paciencia.

A mis amigos que más quiero y admiro, a Elia por darme su amistad, apoyo y consuelo en los momentos más difíciles de mi vida, a Sandy, Laura, Jos, Edgar, por ser grandes compañeros en mi camino y a todos mis amigos y profesores de la Universidad que sin ellos no habría llegado hasta aquí.

CONTENIDO

INTRODUCCIÓN

Objetivo General

Proporcionar al docente que imparte asignaturas relacionadas con la informática y la computación del nivel Bachillerato de la UNAM, un programa de actualización orientado a la programación, abarcando los aspectos didácticos, metodológicos, así como aspectos de actualidad como aplicaciones sobre Internet para conocer y manejar los paradigmas y los diferentes lenguajes de codificación, así como las técnicas de enseñanza de la programación.

Antecedentes del Diplomado

La actualización del personal docente juega un papel muy importante en el ejercicio educativo. Pretende facilitar el proceso enseñanza-aprendizaje incrementando el nivel de calidad académica en el profesor.

La actualización docente significa poner al día las prácticas y métodos de enseñanza.

Por lo tanto, es importante comenzar a desarrollar un programa de capacitación para profesores del área de cómputo para contar con una actualización, ya que es elemental estar en constante preparación.

Sin embargo, otra cosa que es fundamental es tratar de proporcionar a la mayoría del personal docente las herramientas de métodos y técnicas de enseñanza, de manera que pueda trasmitir los conocimientos que posee.

La finalidad de este diplomado, fue la actualización del personal docente en el ámbito de la programación de computadoras, desde una forma sencilla, hasta desarrollar aplicaciones complejas, como aplicaciones que trabajen sobre ambiente WEB.

Aprender a programar es brindar la capacidad de abstracción de problemas, para lograr representarlos mediante un lenguaje de programación. Brindar los conocimientos suficientes para la solución de problemas por medio de diferentes paradigmas tales como programación estructurada o programación orientada a objetos.

El docente conoció y manejó los paradigmas y los principales lenguajes de programación, así como las técnicas de enseñanza necesarias para esta asignatura. También logró entender los conceptos básicos para la creación de aplicaciones sobre Internet.

Mucha gente piensa que estudiar la metodología de la programación es una cosa ardua y aburrida. Se logró que no fuera así y que todo lo que se aprendió a partir de ese momento sea de mucha utilidad para la creación de programas de aplicación general.

Este diplomado persigue que el docente adquiera los conocimientos necesarios para desarrollar programas utilizando los métodos adecuados para conseguir programas funcionales, claros y eficientes con la ayuda de las diferentes herramientas didácticas con las que debe contar el docente

Una buena metodología de la programación pasa por comprender e identificar todos los elementos disponibles para la construcción de un programa.

Lo importante es que se adquirieron conocimientos sólidos de programación, para que fácilmente reconozcan cualquier otro lenguaje con el que trabajemos posteriormente.

Al inicio de este diplomado no se utilizó ningún lenguaje de programación estándar, utilizamos un lenguaje similar al lenguaje humano (pseudocódigo) para explicar instrucciones que se pueden utilizar en la mayoría de ellos. Si se es capaz de hacer un planteamiento correcto utilizando este sistema, posteriormente podemos seleccionar el lenguaje de programación en el que se deseas desarrollar y "traducir" las instrucciones a la sintaxis del lenguaje, todo esto se vera en el capitulo 1.

Se vieron en el capítulo 2 las sentencias básicas de la programación estructurada con ayuda del lenguaje Pascal, le siguió la programación orientada a eventos e interfases de usuario con ayuda de las herramientas con las que cuenta Delphi, en el capítulo 3 el desarrollo de programas orientados a objetos usando como apoyo los diferentes objetos con los que cuenta Java, en el capítulo 4 se encuentra una breve introducción a lo que es la didáctica de la programación y finalmente en el capitulo 5 se encuentran las aplicaciones básicas para desarrollo WEB con ayuda del lenguaje HTML y PHP.

Objetivos Particulares

Solución de problemas y algoritmos

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

- Desarrollar habilidades en el análisis, diseño y construcción de algoritmos, para encontrar la óptima solución a los diferentes problemas que se presentan en orden de complejidad creciente.

Lenguaje de Programación Pascal y Herramientas de Delphi

- Aprender los conceptos de la programación en lenguaje Pascal para desarrollar programas en un lenguaje estructurado como herramienta básica de propósito general, fundamentalmente en el desarrollo de sistemas computacionales.

- Proporcionar los conocimientos básicos, teóricos y prácticos para el diseño y construcción de aplicaciones con interfaz gráfica de usuario, utilizando programación orientada a eventos, con ayuda de las diferentes herramientas con las que cuenta Delphi.

Programación Orientada a Objetos con Java

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

- Conocer las principales características de la programación en Java, los procesos de compilación y ejecución de un programa para añadir diferentes elementos dinámicos en aplicaciones Web.

Didáctica de la programación

- Revisar las estrategias para la enseñanza y evaluación en áreas de cómputo para lograr el desarrollo de conocimientos, habilidades y

destrezas específicas que deben ser dominadas para un aprendizaje efectivo.

- Estimular el mejoramiento continuo de la calidad del docente, mediante la actualización e innovación de los métodos y procedimientos que se utilicen en el desarrollo de las funciones de docencia para crear una mejor institución educativa.

Lenguaje PHP y aplicaciones WEB

- Brindar los conocimientos para la instalación de un servidor Apache utilizando el lenguaje de programación HTML y PHP para la creación de páginas Web dinámicas y otras aplicaciones sobre Internet.

CAPÍTULO 1

SOLUCIÓN DE PROBLEMAS Y ALGORITMOS

1.1. Antecedentes

Para el proceso de desarrollo de sistemas computacionales, se debe considerar una serie de fases o pasos comunes, que generalmente deben seguir todos los programadores.

En este módulo del diplomado se trataron varios temas básicos referentes a la creación de sistemas computacionales, basándose en las diferentes etapas que se deben de tomar en cuenta para lograr un mejor producto que en este caso es el sistema terminado. A todo esto se le denomina Ingeniería de Software[1], en el cual, en lugar de concentrarse solo en la programación del sistema, la ingeniería de sistemas de computadora se concentra en una variedad de elementos, analizando, diseñando y organizando esos elementos para que pueda ser un producto, un servicio o una tecnología.

Los conceptos que se desarrollaron son fundamentales para la solución de problemas. Los problemas, algoritmos y programas forman parte de los estudios de computación.

Un ***problema*** en la mayoría de las ocasiones, suele ser un asunto del que se espera una solución, se puede definir de forma general haciendo uso de identificadores o parámetros, puede contener desde elementos dados, hasta elementos desconocidos. Se puede plantear desde una simple pregunta sobre objetos y estructuras que requieren una explicación y una demostración. Requiere de pensamiento reflexivo y un razonamiento de acuerdo con un conjunto de definiciones, axiomas y reglas, por lo cual esto genera una estructura de pensamiento lógico y simbólico y le da al Ingeniero las herramientas básicas para la innovación y el desarrollo tecnológico.
Para un ingeniero, un problema se puede convertir en parte de un reto que debe resolver con la más óptima de las soluciones, con los mínimos recursos necesarios, al menor tiempo posible, con un bajo costo, pero nunca sin perder de vista el aspecto de la calidad durante el proceso y al final del producto.

1. Ingeniería de Software. Es el estudio de los *principios y metodologías* para desarrollo y mantenimiento de sistemas de software. *Software* es la suma total de los programas de computadora, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo

Se comenzó por plantear algunos elementos importantes que regularmente se presentan en la vida cotidiana, como por ejemplo la solución de un problema como tal, los cuales particularmente se desarrollan con base en lo siguiente pregunta ¿Qué elementos son realmente importantes para trabajar en la solución de un problema?, esto nos adentro sin duda en el hecho de que todo depende de los resultados deseados y de los recursos disponibles, pero indiscriminadamente no podemos trabajar con todos esos elementos al mismo tiempo por la complejidad a la que nos enfrentamos, por lo tanto se trabajó solo con aquellos componentes que presentan mayor significado para lograr el objetivo.

Antes de comenzar de lleno con la aplicación o el desarrollo de cada una de las fases de ingeniería de software, se debe tomar en cuenta inicialmente si el problema que se desea resolver es computable o no, es decir, que si es posible que el problema que se tiene, se puede traducir a un lenguaje computacional, para esto es necesario basarse en la Teoría de la computabilidad, esto a su ves implica el estudio de la *abstracción de datos*, *solución de problemas por medio de algoritmos*, el uso de *pseudocódigo*, etc.

1.2. Teoría de la computabilidad

La teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que pueden ser resueltos con la ayuda de un algoritmo, es decir un procedimiento dado puede ser representado en forma de pasos o que a su ves pueden ser representados en una computadora lógica. Esta teoría tiene mucho que ver con algo que se le conoce como Computabilidad-T o mejor dicho Maquina de Turing, la cual es un modelo computacional con el cual se afirmaba que se podía resolver cualquier problema definiendo si es computable o no.

No todos los problemas pueden ser resueltos, actualmente se conocen muchos problemas indecidibles, por eso es importante primero conocer si el problema al que se enfrenta es descifrable o no, para eso se cuenta con la ayuda de la maquina de Turing.

1.2.1. Máquina de Turing

La máquina de Turing[2] es un modelo computacional introducido por Alan Turing, es un modelo matemático abstracto que formaliza el concepto de algoritmo, son autómatas que pueden ser vistos como mecanismos formales de cómputo, fue el primer modelo teórico de lo que luego sería un computador programable. Como se observa en la Figura 1.1 la Máquina de Turing consta de un cabezal lector/escritor y una cinta infinita en la que el cabezal lee el contenido, borra el contenido anterior y escribe un nuevo valor. Las operaciones que se pueden realizar en esta máquina se limitan a:

- Avanzar el cabezal lector/escritor para la derecha y
- Avanzar el cabezal lector/escritor para la izquierda.

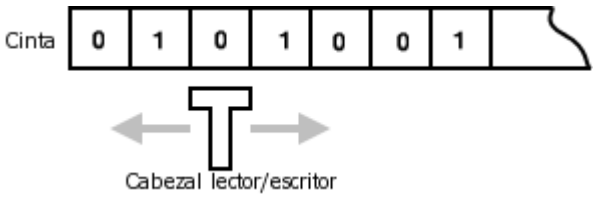


Figura 1.1. Máquina de Turing.

El cómputo es determinado a partir de una tabla de estados de la forma:

(estado, valor) ⟶ (\nuevo estado, \nuevo valor, dirección)

Esta tabla toma como parámetros el estado actual de la máquina y el caracter leído de la cinta, dando la dirección para mover el cabezal, el nuevo estado de la máquina y el valor a ser escrito en la cinta. Con este aparato extremadamente sencillo es posible realizar cualquier cómputo que un computador digital sea capaz de realizar. Mediante este modelo teórico y el análisis de complejidad de algoritmos, fue posible la categorización de problemas computacionales de acuerdo a su comportamiento. De hecho, se puede probar matemáticamente que para cualquier programa de computadora es posible crear una máquina de Turing equivalente. Por lo tanto la maquina de Turing es capaz de procesar ciertas acciones y si esto es posible, entonces quiere decir que el problema es computable, si no lo logra, entonces no lo es. Así que si un proceso no es computable, quiere decir que no existe un algoritmo de solución para ese problema.

1.2.2. Autómatas

Para poder trabajar con todos estos procesos programables o no programables es necesario estudiar un poco a cerca de procesos autómatas y de sus características que lo componen para lograr el desarrollo del análisis de algún problema.

Autómata. Vienen a ser mecanismos formales que procesan un problema computable, realizan derivaciones en gramáticas formales[3] La manera en que las realizan es mediante la noción de *reconocimiento*. Una palabra será generada en una gramática si y sólo si la palabra hace transitar al autómata correspondiente a sus condiciones terminales. Por esto es que los autómatas son *analizadores léxicos* de las gramáticas a que corresponden.

2. Máquina de Turing, introducido por Alan Turing en el trabajo *On computable numbers, with an application to the Entscheidungsproblem,* publicado por la Sociedad Matemática de Londres.
3. Gramática formal. Conjunto de reglas capaces de generar todas las posibilidades combinatorias de ese lenguaje.

Los autómatas se representan por medio de un lenguaje matemático generado por:

- **Símbolos**. Son la mínima unidad de Información por ejemplo:

  a, b, c, <números> , <letras>,

- **Cadenas, Cuerdas, Palabras.** En la representación de autómatas también se forman conjuntos de símbolos los cuales se les conoce como cadenas, cuerdas o palabras.

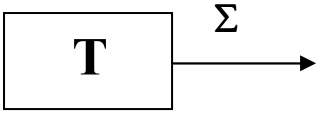
  ab, aaaa, <número> <letra> <letra>

- **(∑) Alfabeto = { a,b }**

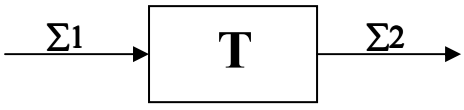
  a, b, aa, ba, bb

- **Aceptadores.** Si entra una cadena a través de un autómata se obtiene como resultado la validez de la cadena.

- **Transductor.** Esto es si se tiene un autómata, de resultado genera cadenas.



- **Traductor.** Si al inicio se tiene una cadenas, se introduce al traductor y de resultado da cadenas en otro alfabeto.



La representación de los autómatas es una quíntupla como se muestra a continuación y la representación gráfica se muestra en la Figura 1.2:

$$A = <Q, \Sigma, Qo, F, \alpha>$$

Q Conjunto de estados.

Σ Alfabeto.

Qo Estado Inicial.

F Conjunto de estados finales.

α Función de transición.

Representación autómata

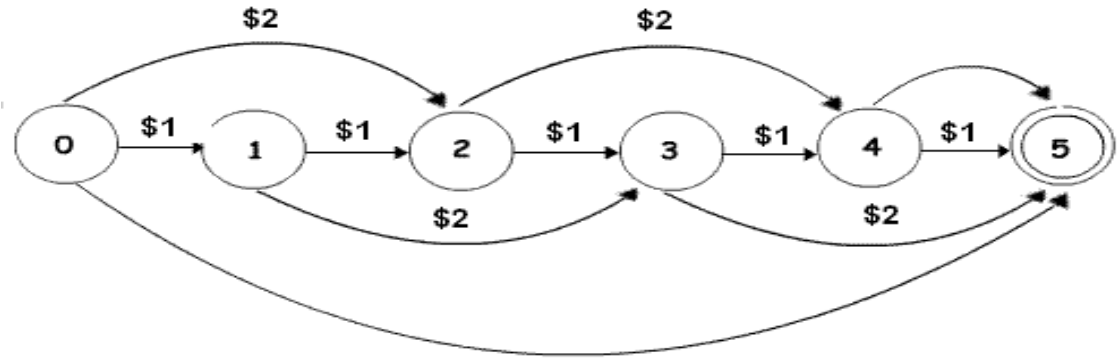


Figura 1.2. Representación autómata.

Q = { 0,1,2,3,4,5 }
Σ = {$1, $2}
Qo = 0
F = {5}

α = (0, $1) ----- 1
(0, $2) ----- 2
(0, $1) ----- 2
(0, $2) ----- 3

En la representación autómata que se mostró anteriormente se puede observar como se grafican los conjuntos de estado, en este caso son círculos con su respectivo número de secuencia, con la ayuda de la información se sabe cual es el estado inicial **Qo** y el estado final **F**, los lazos también contienes el alfabeto con el que los identificamos ∑.

De esta manera se puede saber si un problema es computable y así posteriormente pasar a lo que es la creación de algoritmos para comenzar con el análisis.

1.3. Introducción a los algoritmos

Un Algoritmo es un procedimiento paso a paso que toma cualquier instancia del problema y produce una respuesta correcta, también otra definición seria que es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema.

Los algoritmos se presentan a diario con las diversas vivencias, todo lo que nos rodea es siempre una secuencia de pasos que nos llevan a realizar labores en ocasiones ordenadamente. El algoritmo da la solución genérica a un problema y lo podremos emplear todas las veces que se nos presente ese mismo caso.

Los algoritmos, para llegar a ser tales deben reunir ciertas características. Una de ellas es que los pasos que deben seguirse deben de estar estrictamente descritos, cada acción debe ser precisa, y debe ser general, es decir que pueda ser aplicable a todos los elementos de una misma clase, debe de poseer resolución, en otras palabras, esto quiere decir que el algoritmo deberá de llegar a un resultado específico.

Los algoritmos computacionales deberán cumplir al menos con las siguientes propiedades:

- Cada función debe estar definida.
- Debe establecer claramente lo que debe hacer.
- Cada funcionamiento debe ser eficaz.
- Cada paso debe ser tal que, por lo menos en principio, una persona puede ejecutarlo con sólo papel y lápiz en un tiempo finito.
- El algoritmo debe terminar después de un número finito de pasos.

Ejemplos de algoritmos son:

Ver una Película

1. Buscar el videocasette de la película
2. Si el televisor y la video se encuentran apagados, encenderlos

3. Sacar el video del estuche
4. Introducirlo en la videocasetera
5. Tomar el control del televisor y la video
6. Dirigirme a el sofá
7. Ponerme cómodo
8. Disfrutar la película

Se puede observar como se ha descrito en estos pasos el algoritmo para poder ver una película en la video, este pequeño algoritmo cumple con los requisitos descritos arriba, ya que cada paso precisa un orden y tiene un orden de pasos finitos.

Una vez descubierto un algoritmo para efectuar una tarea, la realización de ésta ya no requiere entender los principios en que se basa dicho algoritmo, pues el proceso se reduce a seguir las instrucciones del mismo. Por ejemplo, podemos hacer una división siguiendo el algoritmo sin entender por qué funciona el algoritmo. La inteligencia requerida para llevar a cabo la tarea está codificada en el algoritmo.

En el ámbito de los ordenadores, los algoritmos se expresan como programas. Los programas son algoritmos codificados con un lenguaje no ambiguo cuya sintaxis y semántica "entiende" el ordenador.

Todo esto quiere decir que inicialmente cuando se requiere planear el desarrollo de un programa, sistema o software en indispensable comenzar por pensar como se podría resolver el problema y por lo tanto es necesario comenzar a desarrollar el algoritmo de solución.

1.3.1 Técnicas de diseño de algoritmos

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. Esta técnica hace que los programas sean más fáciles de escribir, leer y mantener. Algunas de las técnicas más conocidas son: Top Down y Botton UP.

- **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, de solución más sencilla. El problema se descompone en etapas o estructuras jerárquicas, de modo que se puede considerar cada estructura como dos puntos de vista: lo que hace y cómo lo hace.

- **Botton Up.** Esta técnica consiste en partir de los detalles más precisos del algoritmo complementando sucesivamente módulos de mayor complejidad, se recomienda cuando ya se cuenta con experiencia y ya

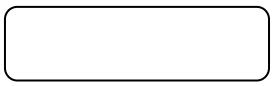
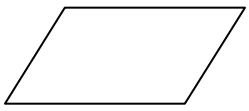
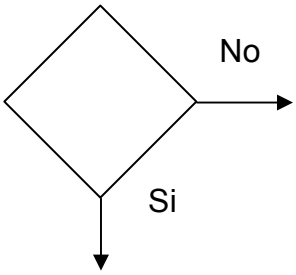
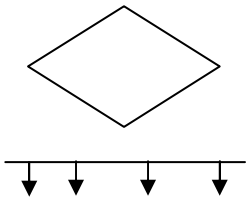
se sabe lo que se va hacer. Este método es lo inverso del anterior. Esta técnica es frecuentemente utilizada para la realización de pruebas a sistemas ya concluidos.

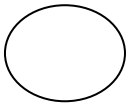
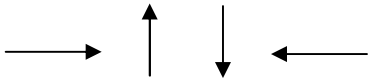
Las técnicas descritas anteriormente permiten un diseño global de algoritmos, pero ocasionalmente pueden desatender detalles específicos de este. Las técnicas más populares de creación de algoritmos son: Diagramas de flujo y Pseudocódigo.

1.3.2. Diagramas de flujo

Se basan en la utilización de diversos símbolos para representar operaciones específicas. Se le llama así por que los símbolos utilizados se conectan por medio de flechas para indicar la secuencia de operación.

Simbología utilizada en los Diagramas de flujo

	**Terminal.** Representa el inicio y el final de un programa.
	**Entrada / Salida.** Cualquier tipo de introducción de datos.
	**Proceso.** Cualquier tipo de operación que pueda originar cambios de valor, formato o posición de la información
	**Decisión.** Indica operaciones lógicas o de comparación entre datos.
	**Decisión Múltiple.** Esta es en función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado

	**Conector.** Sirve para enlazar dos partes de un organigrama.
	**Indicador de dirección o línea de flujo.** Indica el sentido de ejecución de las operaciones.
	Línea Conectora. Sirve de unión entre dos símbolos.

1.3.3. Pseudocódigo

Es un lenguaje de especificación de algoritmos. El uso de tal hace el paso de codificación final (esto es, la traducción a un lenguaje de programación) relativamente fácil.

El pseudocódigo nació como un lenguaje similar al inglés y era un medio representar básicamente las estructuras de control de programación estructurada. Se considera un primer borrador, dado que el pseudocódigo tiene que traducirse posteriormente a un lenguaje de programación. Cabe señalar que este no puede ser ejecutado por una computadora.

La ventaja del pseudocódigo es que en su uso en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico. Es también fácil modificar el pseudocódigo si se descubren errores o anomalías en la lógica del programa, además de todo esto es fácil su traducción a lenguajes como Pascal, COBOL, C, FORTRAN o BASIC.

Como se puede observar ya se cuenta hasta este momento con un resultado que va a permitir continuar con el desarrollo de las etapas de ingeniería de software anteriormente mencionadas, ya se sabe si el problema es computable o no con ayuda de la teoría de la computabilidad y con el apoyo de la máquina de turing, ya se puede iniciar la creación del algoritmo con las diferentes técnicas que se vieron anteriormente y a su ves se puede comenzar a desarrollar el pseudocódigo que se requiere para el desarrollo del sistema,

Pero ahora se tiene que iniciar con el desarrollo de cada uno de los procesos de análisis, diseño e implementación para conseguir que el sistema que se va a construir cuente con los requisitos necesarios para obtener un buen resultado.

Generalmente las etapas utilizadas en el desarrollo de software son:

- Análisis de factibilidad
- Requerimientos de software
- Diseño
- Validación
- Implantación
- Mantenimiento

1.4. Análisis del sistema

El análisis es la primera fase técnica del proceso de ingeniería del software. En este punto se refina la declaración general del ámbito del software en una especificación concreta que se convierte en el fundamento de todas las actividades siguientes.

El análisis debe enfocarse en los dominios de la información, funcional y de comportamiento del problema. Para entender mejor lo que se requiere, se crean modelos, los problemas sufren una partición y se desarrollan representaciones que muestran la esencia de los requisitos y posteriormente los detalles de la implementación.

En muchos casos, no es posible especificar completamente un problema en una etapa tan temprana. Como resultado del análisis se desarrolla la especificación de requisitos del software. La revisión es esencial para asegurarse de que el cliente y el desarrollador tienen el mismo concepto del sistema. Desgraciadamente, incluso con los mejores métodos, la cuestión es que el problema sigue cambiando.

El análisis del sistema se lleva a cabo con los siguientes objetivos:

1. Identificar la necesidad.
2. Evaluar el concepto del sistema para establecer la viabilidad.
3. Realizar un análisis técnico y económico.
4. Asignar funciones al hardware, software, bases de datos y otros elementos del sistema.
5. Establecer las restricciones del presupuesto y planificación temporal.
6. Crear una definición de sistema que forme el fundamento de todo el trabajo de ingeniería subsiguiente. Se requiere un gran dominio del hardware y del software para conseguir con éxito los objetivos mencionados anteriormente.

El primer paso de análisis del sistema afecta a la identificación de necesidades, la intención es entender los objetivos del producto y definir las metas necesarias para alcanzar esos objetivos.

Durante el análisis técnico, se evalúan los principios técnicos del sistema al mismo tiempo que se recoge información adicional sobre el rendimiento,

fiabilidad, características de mantenimiento y productividad. En algunos casos esta fase de análisis también incluye una cierta cantidad de investigación y diseño. Los resultados obtenidos del análisis técnico forman la base para otra decisión de “continuar /abandonar” sobre el sistema.

Para lograr el éxito en el desarrollo de software es esencial una comprensión total de los requisitos del software. No importa lo bien diseñado o codificado que este un programa si no se ha analizado correctamente.

La evaluación del problema y la síntesis de la solución es una de las áreas principales del análisis, el proceso debe ir desde la información esencial hasta el detalle de implementación.

1.4.1. Lenguaje Unificado de Modelado UML

Para comprender el tema de análisis, se baso principalmente en una herramienta que se conoce como el Lenguaje Unificado de Modelado (UML). Luego de que se investigaron las diferentes características de este tema, se encontró que es muy extenso, así que sólo se logro desarrollar algunas de sus características importantes del mismo que permita complementar el contenido del tema. Se comenzó por describir que es UML, el cual un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software[4]. Se usa principalmente para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir, capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. Se debe tomar en cuenta que UML no es un lenguaje de programación.

Objetivos de UML

- UML es un lenguaje de modelado de propósito general que puede usar todos los modeladores. No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática.
- No pretende ser un método de desarrollo completo. No incluye un proceso de desarrollo paso a paso. UML incluye todos los conceptos que se consideran necesarios para utilizar un proceso moderno iterativo, basado en construir una sólida arquitectura para resolver requisitos dirigidos por casos de uso.
- Pretende ser tan simple como sea posible pero manteniendo la capacidad de modelar toda la gama de sistemas que se necesita construir.
- Necesita ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la

concurrencia y distribución, así como también los mecanismos de la ingeniería de software, como son la encapsulación y componentes.
- Debe ser un lenguaje universal, como cualquier lenguaje de propósito general.
- Pretende imponer un estándar mundial.

Como se observa, abarca todo el proceso de vida de desarrollo del software[5].

Una de las primeras etapas que consideramos la más importantes es la de "análisis de requerimientos", Un requerimiento es una característica necesaria que deberá poseer el nuevo sistema, por otra parte, la determinación de requerimientos es el estudio de un sistema para comprender cómo trabaja y dónde es necesario efectuar mejoras, es donde se lleva a cabo el proceso de descubrir, analizar, escribir y verificar los servicios y restricciones del sistema, nos dimos cuenta que su importancia estriba en que, de la definición de los requerimientos dependerá la definición de las etapas subsecuentes del desarrollo de software, es decir, que si no se descubren los requerimientos que se encuentran en el ambiente del sistema ó son encontrados en una etapa avanzada del desarrollo del sistema, se tendrá que retroceder nuevamente a la etapa de requerimientos y esto provocaría cambios en el sistema y consecuentemente retraso en la entrega del mismo. Por lo tanto en esta etapa entramos con la pregunta de ¿Qué hace el sistema? (Algorítmica) y ¿Cómo lo hace? (Lógica).

Comenzamos identificando diferentes elementos para el diseño con ayuda de los distintos tipos de diagramas que existen como son los[6]

- Diagramas de casos de usos
- Diagramas de secuencia
- Diagrama general de casos de uso
- Diagramas de transición de estados

Más adelante se vio en que consisten todos y cada uno de ellos para tener una idea de cual es el más conveniente aplicar según las necesidades.

4. G. Booch, J. Rumbaugh y I. Jacobson, *El Lenguaje Unificado de Modelado*. p.25.
5. I. Jacobson, G. Booch, J. Rumbaugh. *El proceso Unificado de Desarrollo*. p.69.
6. E. Hernandez, J. Hernadez, C. Lizandra. *C++ Estandar*. p.124.

1.4.2. Diagramas de caso de uso

En el estudio de casos de uso se observó que es una técnica para capturar información, de cómo trabaja un sistema, o de cómo se desea que trabaje. No pertenece estrictamente al enfoque orientado a objeto, es una técnica para captura de requisitos. Es una actividad de orden genérico que engloba varios casos.

Los Casos de Uso describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista del usuario. Permiten definir los límites del sistema y las relaciones entre el sistema y el entorno. Son descripciones de la funcionalidad del sistema independientes de la implementación. Y sobre todo están basados en el lenguaje natural, es decir, es accesible por los usuarios. Así que en términos generales son las condiciones que se tiene que cumplir para que el proceso se de, por ejemplo es como el dinero en una transacción, esto es regla de negocios.

Por otra parte en estos diagramas manejan los Actores, los cueles son en este caso:

- Actores principales: personas que usan el sistema.
- Actores secundarios: personas que mantienen o administran el sistema.
- Material externo: dispositivos materiales imprescindibles que forman parte del ámbito de la aplicación y deben ser utilizados.
- Otros sistemas: sistemas con los que el sistema interactúa.

Los casos de uso intervienen durante todo el ciclo de vida. El proceso de desarrollo estará dirigido por los casos de uso. Un escenario es una instancia de un caso de uso. (Figura 1.3).

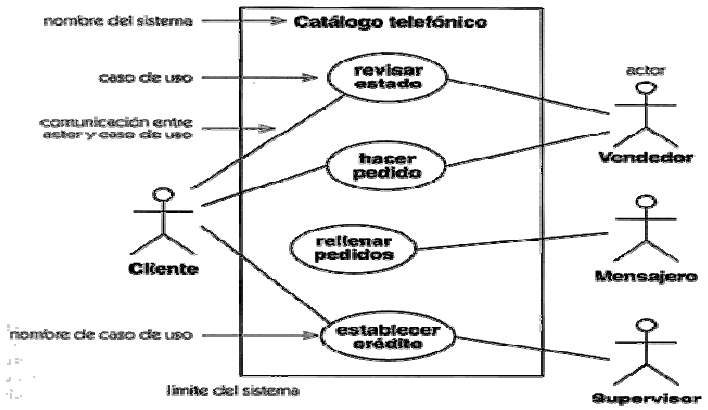


Figura 1.3. Diagrama de caso de uso.

1.4.3. Diagramas de secuencia

Para este caso los diagramas de secuencia son los que muestran las interacciones entre los objetos organizados en una secuencia temporal. En particular muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados.

Representa una interacción, un conjunto de comunicaciones entre objetos organizadas visualmente por orden temporal. A diferencia de los diagramas de colaboración, los diagramas de secuencia incluyen secuencias temporales pero no incluyen las relaciones entre objetos. Pueden existir de forma de descriptor (describiendo todos los posibles escenarios) y en forma de instancia (describiendo un escenario real).

Dentro del conjunto de mensajes representados dispuestos en una secuencia temporal, cada rol en la secuencia se muestra como una línea de vida, es decir, una línea vertical que representa el rol durante cierto plazo de tiempo, con la interacción completa. Los mensajes se muestran como flechas entre líneas de vida. Un diagrama de secuencia puede mostrar un escenario, es decir, una historia individual de transacción. Un uso de un diagrama de secuencia es mostrar la secuencia del comportamiento de un caso de uso.

Un diálogo de secuencia posee dos dimensiones: la vertical representa el tiempo, la horizontal representa los objetos que participan en la interacción. En general, el tiempo avanza hacia abajo dentro de la página (se pueden invertir los ejes si se desea). Con frecuencia sólo son importantes las secuencias de mensajes pero en aplicaciones de tiempo real el eje temporal puede ser una métrica. La ordenación horizontal de los objetos no tiene ningún significado.

Como se muestra en la Figura 1.4, cada objeto representa una columna distinta, se pone un símbolo de objeto al final de la flecha que representa el mensaje que ha creado el objeto; está situada en el punto vertical que denota el instante en que se crea el objeto. Esta se conoce como línea de vide del objeto. Se pone una X grande en el punto en que deja de existir el objeto o en el punto en que el objeto se destruye a sí mismo. Para el periodo durante el cual esté activo el objeto, la línea de vida se amplía para ser una línea doble continua. Si el objeto se llama a sí mismo, entonces se superpone otra copia de la doble línea para mostrar la doble activación. El orden relativo de los objetos no tiene significado aún cuando resulta útil organizarlos de modo que se minimice la distancia de las flechas.

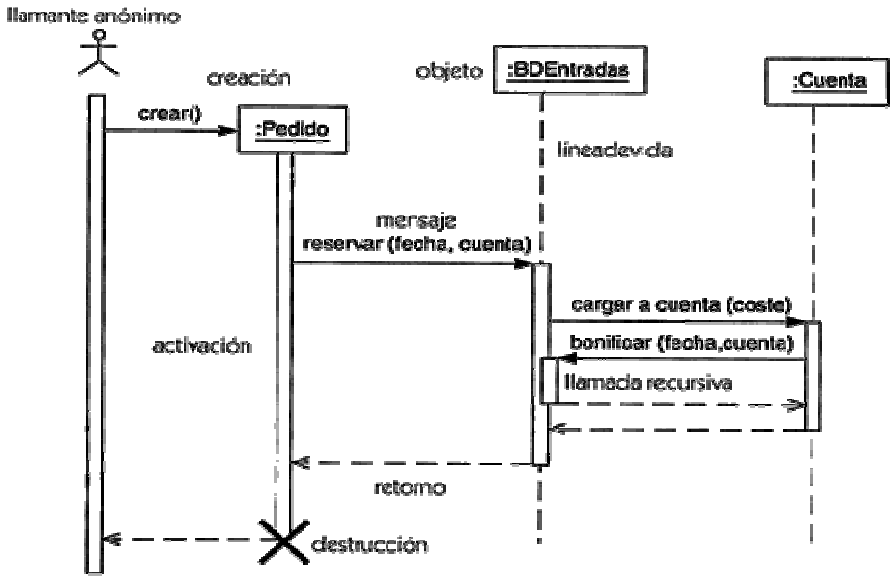


Figura 1.4. Diagrama de secuencia.

Existen otros tipos de diagramas, como diagramas de componentes, despliegue, paquetes, entre otros, los cuales todos tienen una técnica diferente que podemos usar como herramienta para el diseño de nuestro software, pero por el momento sólo mencionamos los anteriores.

1.5. Diseño del sistema

El diseño de software se sitúa en el núcleo del proceso de ingeniería del software y se aplica independientemente del paradigma de desarrollo utilizado. Suponiendo que se hayan analizado y especificado los requisitos del sistema, el diseño es la primera de las tres actividades técnicas diseño, codificación y prueba, necesarias para construir y verificar el software.

Cada uno de los elementos del modelo de análisis proporciona información necesaria para crear un modelo de diseño, esto se muestra en la figura

El diseño es un proceso y un modelo a la vez, es un conjunto de pasos repetitivos que permiten describir todos los aspectos del sistema a construir.

Se sabe que para un buen diseño se requiere tomar en cuenta muchos aspectos, así que cabe mencionar que solo se desarrollaron los principales.

1.5.1. Diseño conceptual

El diseño conceptual se refiere a la definición de las funciones de los diferentes módulos, en forma algorítmica. Esto proporciona una visión general de si el sistema sobre el cual se está trabajando es factible o eficiente o no lo es. Durante el diseño conceptual como se dijo anteriormente se modelan los requerimientos del usuario, para adaptar el modelo a la migración.

Para identificar a los usuarios o actores del sistema como son llamados en UML se debe hacer un análisis de las personas que utilizan en la actualidad el sistema y papeles que juegan en su interacción con el mismo.

A pesar de las ventajas que ofrece, en determinadas circunstancias se observa que es necesario adecuar las herramientas a las necesidades que se requieren o bien crear una que responda a estas necesidades.

- Identificar la organización definiendo sus necesidades de gestión e información.
- Identificar las relaciones entre las necesidades de los clientes y los documentos.
- Identificar los flujos documentales.
- Analizar la información contenida en los documentos diferenciando sus atributos: forma, tipo y material.
- Identificar las características establecidas en la información contenida en los documentos.
- Definir los campos
- Establecer las opciones de búsqueda.
- Desarrollar el mapa conceptual dibujando el diagrama con las relaciones entre la organización, los documentos y la información.
- Revisar el esquema conceptual con el usuario.
- Diseñar aplicaciones informáticas nuevas o evaluamos las del cliente para su modificación.

1.5.2. Construcción del software

Otra de las etapas del Diseño es la de construcción de software, es decir, determinar el software o el compilador que se requiere para programar nuestro sistema y que cubra todas las necesidades. Las diferentes metodologías y estándares relacionados con el desarrollo y construcción de software por lo regular buscan mantener altos niveles de confiabilidad y control de la solución informática, la seguridad informática y sus principios de diseño.

En esta fase se precisa la estructura de cada uno de los componentes de software del sistema, descomponiendo cada uno en tratamientos y datos elementales.

Se definen los algoritmos y estructuras de datos en lenguaje semi-formal. En esta fase se preparan los procedimientos para las pruebas por unidad del sistema.

Codificación

Esta fase tiene como objetivo traducir el diseño detallado de cada uno de los componentes de software al lenguaje de programación o parámetros de configuración seleccionados. Se refiere a la obtención de un programa definitivo que pueda ser comprensible para la máquina. Incluye una etapa que se reconoce como compilación.

1.5.3. Documentación del sistema

La documentación de sistemas es el conjunto de información que explica qué hacen los sistemas, cómo lo hacen y para quién lo hacen. La documentación consiste en material que explica las características técnicas y la operación de un sistema. Es esencial para proporcionar entendimiento de un sistema a quien lo vaya a usar para mantenerlo, para permitir auditoria del sistema y para enseñar a los usuarios como interactuar con el sistema y a los usuarios como hacerlo funcionar.

Existen varios tipos de documentación. La de programas, que explica la lógica de un programa e incluye descripciones, diagramas de flujo, listados de programas y otros documentos; la del usuarios en forma general la naturaleza y capacidades del sistema y cómo usarlo. La documentación adecuada y completa, de una aplicación que se desea implantar, mantener y actualizar en forma satisfactoria, es esencial en cualquier sistema de información.

1.5.4. Pruebas operacionales y funcionales del sistema

El objetivo de esta fase es garantizar que cada componente esté conforme con su descripción en el diseño detallado. La buena ejecución de esta fase permite disminuir la carga de trabajo de la integración en la medida en que esta se efectúa con componentes sin defectos.

En esta etapa los componentes del sistema son ensamblados progresivamente, según la estrategia determinada en la fase de diseño preliminar. En cada etapa de la integración es verificado el buen funcionamiento de los componentes integrados en la nueva etapa, con respecto al conjunto de componentes integrados en las etapas precedentes.

Aquí se puede ser muy crítico, en la medida que los eventuales problemas descubiertos son en general difíciles de corregir. Para evitar atrasos y mayores trabajos no considerados, la integración requiere una preparación rigurosa, en que defina la estrategia de integración, las responsabilidades, una política de punto de no retorno

En este caso lo que se observo es que las pruebas operacionales son las que se encargan de verificar si nuestro sistema esta realizando correctamente las funciones que nosotros requerimos para obtener los resultados deseados. Tiene por objetivo comprobar que el sistema cumple con sus especificaciones.

1.6. Implementación del Sistema

La fase de implementación del sistema se refiere a la puesta en marcha del software ya terminado. Es decir instalar y probar físicamente los resultados que da el sistema realizado. Esto implica la instalación de los diferentes dispositivos necesarios para lograrlo (PC's, Servidores, terminales, etc.). Por lo tanto en esta fase se hacen las pruebas finales del sistema como cambios o modificaciones.

1.7. Mantenimiento del Sistema

Esta fase se deja al final por la razón de que un mantenimiento a un sistema se realiza a petición de los usuarios por detección de algunos problemas del sistema o por la necesidad de una mejora del mismo. En este proceso se realiza el registro de las peticiones de mantenimiento recibidas con el fin de llevar el control de las mismas para poder sacar estadísticas de los cambios realizados, por lo tanto es recomendable llevar un catalogo de mantenimiento sobre los sistemas de información. Existen varios tipos de mantenimientos como correctivo, evolutivo, adaptativo, perfectivo por mencionar algunos.

La fase de mantenimiento vuelve aplicar los pasos de las fases de definición y de desarrollo, pero en el contexto del software ya existente. Durante la fase de mantenimiento se encuentran cuatro tipos de cambios:

- ***Corrección***. Incluso llevando a cabo las mejores actividades de garantía de calidad, es muy probable que se descubran defectos en el software, este mantenimiento modifica el software para corregir los defectos.
- ***Adaptación.*** Con el paso del tiempo es probable que cambie el entorno original, por ejemplo, el sistema operativo, el hardware, etc. Este tipo de mantenimiento produce modificación en el software para acomodarlo a los cambios de su entorno externo.
- ***Mejora.*** Conforme se utilice el software, se pueden descubrir funciones adicionales que van a producir beneficios, este tipo de mantenimiento lleva al software más allá de sus requisitos funcionales originales.
- ***Prevención***. El software se puede deteriorar debido al cambio, así que este tipo de mantenimiento, se debe conducir para permitir que el software sirva para las necesidades, la finalidad es que se pueda corregir, adaptar y mejorar más fácilmente.

1.8. Relevancia del la Ingeniería de Software

Hoy día existe gran relevancia sobre la necesidad de un enfoque más disciplinado para el desarrollo de sistemas de computación que el que se ha utilizado hasta ahora. Este enfoque está comprendido en la fase de Ingeniería de Software.

Tienen como elemento común tratar de construir sistemas utilizando principios de ingeniería para el desarrollo de estos, consiste en el establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico que sea fiable y funcione de manera eficiente sobre máquinas reales.

La Ingeniería de Software es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas (eficaces en costo o económicas) a los problemas de desarrollo de software, es decir, permite elaborar consistentemente productos correctos, utilizables.

El proceso se define como un conjunto de etapas parcialmente ordenadas con la intención de logra un objetivo, en este caso, la obtención de un producto de software de calidad.

El análisis y estudio de los algoritmos es una disciplina de las ciencias de la computación, y en la mayoría de los casos su estudio es completamente abstracto sin usar ningún tipo de lenguaje de programación ni cualquier otra implementación; por eso, en ese sentido, comparte las características de las disciplinas matemáticas.

En la ciencia de la computación y en la programación, los algoritmos son más importantes que los lenguajes de programación o las computadoras. Un lenguaje de programación es tan solo un medio para expresar un algoritmo y una computadora es solo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente. Dada la importancia del algoritmo en la ciencia de la computación, un aspecto muy importante será el diseño de algoritmos. El diseño de la mayoría de los algoritmos requiere creatividad y conocimientos profundos de la técnica de la programación. En esencia, la solución de un problema se puede expresar mediante un algoritmo.

Finalmente durante el desarrollo de este tema es muy significativo el estudio de la etapa de análisis en el desarrollo de software ya que es donde se lleva a cabo el proceso de descubrir, analizar, escribir y verificar los servicios y restricciones del sistema. Su importancia estriba en que, de la definición de los requerimientos dependerá la definición de las etapas subsecuentes del desarrollo de sistemas, es decir, que si no se descubren los requerimientos que se encuentran en el ambiente del sistema ó son encontrados en una etapa avanzada del desarrollo del sistema, se tendrá que retroceder nuevamente a la

etapa de requerimientos y esto provocaría cambios en el sistema y consecuentemente retraso en la entrega.

Un caso peor, es que no se encontraran y especificarán todos los requerimientos en un proceso de desarrollo de software, lo cual produciría la entrega de un producto incompleto o poco funcional.

Por ello la necesidad de conocer los diferentes elementos que se deben tomar en cuenta para lograr los objetivos que se plantean desde un inicio los cuales en su totalidad es obtener un sistema funcional y así lograr los resultados deseados.

Se descubrió que es necesario comenzar con un buen análisis de lo que realmente se necesita antes de comenzar a programar un sistema y que esto ayuda a evitar errores que a veces pueden ser irremediables.

Por consecuencia también se logró obtener un panorama general de las herramientas que se encuentran en nuestro entorno, las cuales pueden ayudar a entender todo el desarrollo que se debe de llevar para la realización de cualquier sistema de software. Todo esto con ayuda de procedimientos como lo que brinda el lenguaje de modelado UML

Es imprescindible comprender que si se logra hacer un buen análisis detallado de lo que se busca como resultado, lograremos grandes beneficios al final, ya que la mayoría de las veces va a ser realmente satisfactorio y con la mínima posibilidad de errores o modificaciones.

CAPÍTULO 2

LENGUAJE DE PROGRAMACIÓN PASCAL Y HERRAMIENTAS DE DELPHI

2.1. Antecedentes

En el segundo módulo del diplomado de actualización se estudió el lenguaje de programación Pascal el cual es un lenguaje de programación de alto nivel, se nombró a honor de Blaise Pascal[7], se creó en la década de los 70 con el objetivo de disponer de un lenguaje de alto nivel y propósito general, se utiliza para gran diversidad de aplicaciones, esta orientado hacia los nuevos conceptos de programación, desarrollado por el profesor suizo Niklaus Wirth como un lenguaje para la enseñar la programación de modo disciplinado, fácil de aprender y con la complejidad suficiente para permitir una correcta preparación de los programadores futuros.

Una versión preliminar del lenguaje apareció en 1968 y el primer compilador[8] totalmente completo apareció a finales de 1970.

Desde entonces muchos compiladores han sido construidos y están disponibles para diferentes máquinas. De entre todas ellas Turbo Pascal de Borland y su sucesor Delphi, estos son sin duda la versión más importante entre los compiladores desarrollados, ofreciéndonos un entorno operativo de programación en la que se integran el mismo editor de programa, el compilador y un intérprete, perfectamente ajustados para desarrollar programas en Pascal.

El lenguaje estándar presenta una serie de características que lo hacen el lenguaje perfecto para aquellas personas que inician en la programación:

- Es excelente para el aprendizaje de la programación.
- Lenguaje de propósito general, ya que se puede aplicar a gran diversidad de aplicaciones.
- Utilización de procedimiento (programación modular).

7. *Blaise Pascal*. Fue un matemático, físico y filosófico religioso francés, sus contribuciones incluyen calculadoras mecánicas, teoría de la probabilidad, y el famoso triangulo de pascal.
8. *Compilador* es un programa que hace posible la traducción de un lenguaje de programación al medio que solo entiende el ordenador.

- Es un lenguaje estructurado, se utilizan secuencias de control de bifurcación y bucles(if, for, while, repeat) sin necesidad de la famosa instrucción GOTO tan utilizada en muchos lenguajes como BASIC.
- Soporta la recursividad, propiedad que tienen los procedimientos para llamarse a sí mismo.
- Cuenta con tipos de datos simples y estructurados, así como definidos por el usuario.
- Tiene posibilidad de trabajar con punteros (variables dinámicas), de este modo permite definir nuestras propias estructuras de datos dinámicas (lista, pilas, colas, etc).
- No es sensible al tamaño (mayúsculas, minúsculas no hacen diferencia en identificadores).
- Fuertemente tipado. Se cuanta con muchos tipos de datos básicos y se pueden crear tipos nuevos de manera muy sencilla.
- Esta diseñado considerando los conceptos de programación estructurada.

Por otra parte Delphi es una herramienta que se tomo en cuenta para lograr desarrollar sistemas computacionales pero en una forma grafica o visual, basándose en los conocimientos adquiridos en pascal ya que Visual Delphi esta basado principalmente en el lenguaje Pascal conocido en Delphi como Object Pascal. Todo esto se logró a través del desarrollo de este módulo.

2.2. Lenguaje de programación Pascal

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

2.2.1. Tipos de datos básicos

Los diferentes objetos de información que Pascal utiliza para trabajar se conocen con el nombre de datos. Cada uno de estos objetos tendrá un tipo de datos diferente acorde con su información. Según del tipo de información de que se trate, podría ser un número, una letra, etc.

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

- Detectar errores de operaciones en programas.
- Determinar como ejecutar las operaciones.
- Determinar el rango de valores que puede tomar una variable
- Reservar en memoria el espacio correcto para cada tipo de datos.

En pascal todos los datos que se utilizan deben de tener sus tipos declarados previamente.

Los tipos básicos de Pascal son los que se muestran en la Tabla 2.1. 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 de 2 bytes	0 a 65535
Longint	Entero largo	-2*10^31 y 2*10^31
Char	Carácter	Un carácter ASCII
String	Cadena	Cadena de texto ASCII
Bolean	Booleano	Verdadero o falso
Real	Real o de punto flotante	-2.9*10^39 y 1.7*10^38

Tabla 2.1. Tipos Básicos de Pascal.

2.2.2. Estructura de un programa en Pascal

Program nombre (input, output) Uses Nombres de unidad;	Nombre del programa, es opcional Indicación de que bibliotecas precompiladas (Unidades) se cargaran
Const Declaración de constantes;	Opcional.
Type Declaración de tipos de datos;	Opcional.
Var Declaración de variables;	Opcional, pero casi siempre aparece.

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.

Esta es la descripción básica de un programa en Pascal, los detalles de cada parte se analizaron más adelante conforme se requirieron.

2.2.3. Identificadores

Las variables y las funciones son elementos esenciales para la creación de programas, pero para manejarlos es preciso que se le asigne un nombre, a estos “nombres” para variables y funciones les llama identificadores. En pascal se crean utilizando sencillas reglas:

- Empiezan con una letra o el símbolo de guión bajo “_”.
- Después pueden contener letras, números o guión bajo en cualquier orden con una longitud igual a cero o mas caracteres.

Por ejemplo: Correctos – Nombre, _Sueldo, C33.

Incorrectos – 3Intento, $valor, Valor futuro.

2.2.4. Cuando utilizar constantes

Una de las razones más importante para utilizar constantes de expresión son: reconfiguración y documentación. Si utilizamos un valor en un programa que se repita en numerosas ocasiones será interesante utilizar una constante, ya que si en el futuro se quiere cambiar ese valor por otro, solo se cambiar el valor de la constante, afectando a todo el programa.

Deben ser declaradas antes de su utilización bajo la palabra reservada CONST, pueden ser de cualquier tipo de datos, y su valor debe ser asignado mediante el signo igual.

La sintaxis es de la siguiente forma:

CONST

Identificador = valor o expresión;

La expresión podrá ser una operación matemática entre dos valores cualesquiera, o entre los valores almacenados en otras constantes o variables.

2.2.5. Cuando usar variables

Las variables son objetos de un programa cuyo contenido puede variar durante su toda su ejecución. El valor que queremos almacenar en cada una de ellas se llevará a cabo mediante la sentencia de asignación, No se debe confundir el valor que almacena una variable con el tipo de datos que puede almacenar.

Una variable es en realidad una posición de memoria con nombre. A este identificador por el cual conocemos esta posición de memoria se le llama nombre de la variable, y el valor almacenado en ella se denomina valor de la variable.

Todas las variables que se utilicen en el programa deben de estar declaradas previamente en la parte de declaración de variables, parte VAR, de l programa.

La utilización de las variables, depende de la necesidad que se tiene para almacenar datos, en este caso se debe de utilizar variables y no constantes cuando nuestros datos dependen de una entrada externa o dada por el usuario.

Las declaraciones se deben realizar de la siguiente forma:

VAR
Identificador: tipo de datos;
VAR
Listavar1: tipo1;
Listavar2: tipo2;

Ejemplo:

VAR
Edad: Integer;
Apellidos: String;

Es recomendable y una buena práctica de la programación utilizar nombres de variables significativas, que sugieran lo que en ellas representan, es decir, que valor va albergar, de forma que haga el programa más legible y comprensible, así como utilizar comentarios que indiquen para que sirve cada una de las variables.

2.2.6. Operadores

Los operadores se clasifican de acuerdo al tipo devuelto, existen operadores aritméticos, lógicos, de comparación, de asignación y de manejo de bits.

- **Operadores aritméticos:** Toman como operandos números, el valor devuelto es también un número. Los operandos y el resultado pueden ser cualquier variedad de enteros o reales.

Operador	Significado	Ejemplo	Resultado
+	Suma	A + B	Suma A y B
-	Resta	A –B	Diferencia de A y B
*	Multiplica	A * B	Producto a por B
/	División	A / B	Cociente A por B
div	División entera	A div B	Cociente entero A por B

mod	Módulo	A mod B	Resto cociente de A por B
shl	Desplazamiento a izquierda	A shl B	Desplaza A izquierda B bits
shr	Desplazamiento a derecha	A shr B	Desplaza A derecha B bits

Regla de evaluación de expresiones:

- Todas las subexpresiones entre paréntesis primero. Las subexpresiones con paréntesis anidados se evalúan de dentro-afuera; el paréntesis más interno se evalúa primero.
- *Prioridad de operaciones*. Dentro de una misma expresión o subexpresión, los operadores se evalúan en el siguiente orden:

  * , / , div, mod (primero)

  +, - *(segundo)*

- *Regla asociativa izquierda.* Los operadores en una misma expresión o subexpresión con igual nivel de prioridad (tal como * y /) se evalúan de izquierda a derecha.

- **Operadores de comparación:** dos elementos o datos pueden compararse. El resultado de tal comparación será un valor booleano TRUE o FALSE. Las comparaciones pueden hacerse entre variables, constantes y funciones, siempre y cuando correspondan el mismo tipo (Tabla 2.2).

Operador	Operación
=	Igual
<>	Diferente
<	Menor a
>	Mayor a
<=	Menor o igual a
>=	Mayor o igual a

Tabla 2.2. Operadores de Comparación.

- **Operadores booleanos**: se utilizan para relacionar dos o más condiciones, construyendo una expresión condicional.

Por ejemplo:

(A < = B), (C> D) Son dos condiciones.
(A < = B) OR (C > D) Es una expresión condicional.

Los operadores booleanos son:

NOT	Negación, “no”
AND	Conjunción, “y”
OR	Disyuntiva, “o”

Tablas de Verdad. Son un importante auxiliar para determinar el posible resultado de una expresión condicional.

- Tabla de verdad del *NOT*

NOT FALSE	TRUE
NOT TRUE	FALSE

- Tabla de verdad del *AND*

FALSE AND FALSE	FALSE
FALSE AND TRUE	FALSE
TRUE AND FALSE	FALSE
TRUE AND TRUE	TRUE

- Tabla de verdad del *OR*

FALSE OR FALSE	FALSE
FALSE OR TRUE	TRUE
TRUE OR FALSE	TRUE
TRUE OR TRUE	TRUE

El resultado es verdadero cuando alguna condición es verdadera

Precedencia. La elaboración de las operaciones en PASCAL y en los demás lenguajes, sigue ciertas normas de orden. Ya que existen operaciones que se ejecutan primero en una expresión y otras que se ejecutan posteriormente.

Por ejemplo:

*A:= 5*8+3:*
A= 43, se hace 58 y luego +3
*(*y no A=55, suponiendo 8+3 y luego *5*)*

2.2.7. El uso de arreglos

En el manejo de los datos, para la solución de un problema a través de un programa, se hace una abstracción de la realidad para representar la información de una manera ordenada.

El lenguaje Pascal permite implementar un conjunto de abstracciones que son comunes en la mayoría de los problemas de procesamiento de datos, entre ellos las matrices, los conjuntos, las listas de elementos de información, etc. Como se había presentado anteriormente, una de las ventajas que tiene Pascal, con nuevo tipo de variables, que en la mayoría de los casos es definitivo en términos de un tipo primitivo (INTEGER, REAL, CHAR BOOLEAN).

Una estructura de datos es una conformación ordenado de los datos dentro de la memoria de la computadora, que permite un manejo fácil y rápido de la información.

El programador puede determinar la estructura de datos a través de esos tipos, pudiendo definir estructuras, implementando un manejo de datos tan complejo como se desee.

En Pascal los tipos estructurados implementados son:

- Arreglos
- Récord
- Conjuntos
- Archivos

El *arreglo* es una estructura estática de datos, fija y homogénea o sea:

- El número de elementos de los que consta no variará a lo largo de la ejecución del programa.
- Los elementos que constituye la estructura son del mismo tipo.

Se le llama estructura estática porque, una vez que se ha definido el total de datos que agrupará, no se puede aumentar ni disminuir ese tamaño.

Un arreglo gráficamente se representaría de la siguiente forma:

8.04	9.58	7.75	6.89	5.42	9.76

La sintaxis es:

TYPE
<Tipo de arreglo> = [<Tipo de indice>] OF <tipo>;
VAR
< variable arreglo> : <tipo arreglo>;

Donde:

<tipo arreglo> es un identificador cualquiera.
<tipo> es cualquier tipo, primitivo o determinado ANTERIORMENTE.
<variable arreglo> es un identificador cualquiera.

<tipo de índice> puede ser CHAR, BOOLEAN, INTEGER, pero no REAL.

2.3. 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.

2.3.1. 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 2.1:

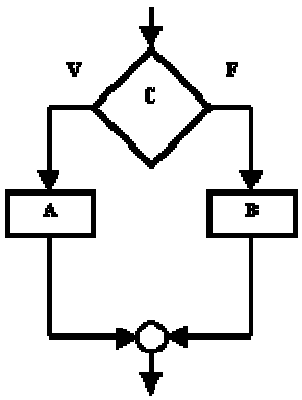


Figura 2.1. Diagrama de Flujo Sentencia IF.

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: Energía');
END.
```

2.3.2. 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 2.2:

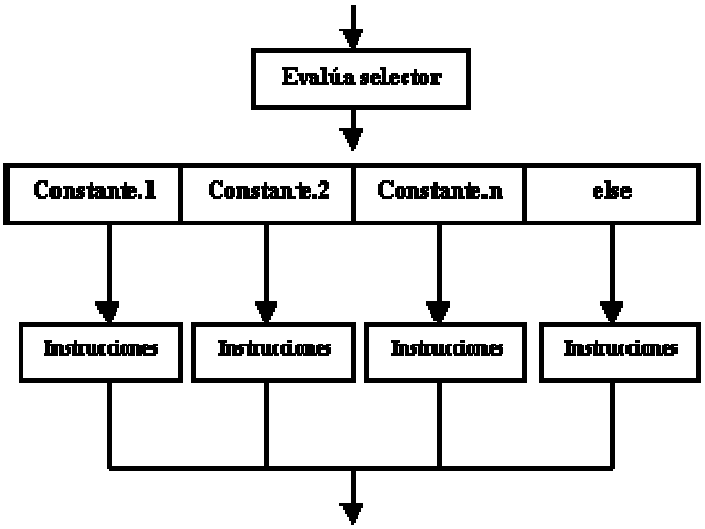


Figura 2.2. Diagrama de Flujo Sentencia CASE.

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**

2.3.3. 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:

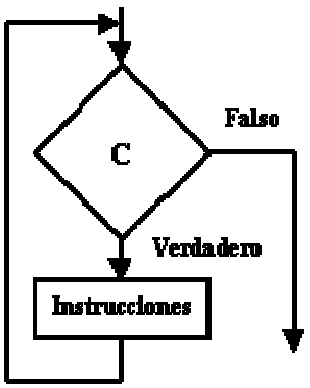


Figura 2.3. Diagrama de Flujo Sentencia WHILE.

***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.
```

2.3.4. 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 2.4:

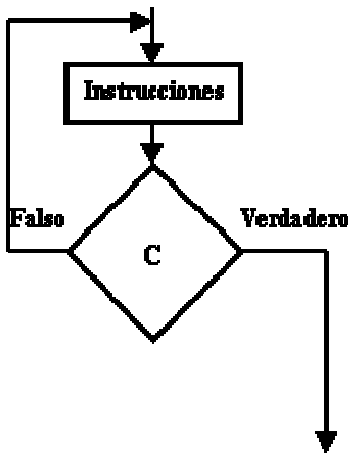


Figura 2.4. Diagrama de Flujo Sentencia REPEAT.

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.
```

2.3.5. Sentencia FOR

La sentencia For repite la ejecución de una o varias sentencias un número 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 actractivo 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.**

2.4. Funciones de entrada y salida

Leer

- **Read ( );** Este se usa cuando leemos archivos binarios.
- **Readln( );** Este se usa de entrada del teclado.

Escribir

- **Write ( );** While (‘Hola’);
  While (‘¿Cómo estas?’);
  Nota: Cuando no le decimos en donde escribir, escribe
  En la salida estandar (monitor)
  Con Writeln : Hola, ¿Cómo estas?

2.5. Como manejar archivos en Pascal

Otro constructor de tipos específico de Pascal es el tipo *file*. Los tipos de archivo representan archivos de disco físicos, lo cual es, ciertamente, una peculiaridad del lenguaje 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, tal como se presenta en la siguiente figura.

Puede definir un nuevo tipo de archivo de debe seguir una estructura como se muestra a continuación:

Nombre Archivo: File of <Tipo>
type
IntFile = file of Integer;

Entonces se puede abrir un archivo físico asociado a esta estructura y escribir valores enteros en él, o leer los valores del mismo.

Para poder usar un archivo físico es necesario asociarlo a la variable con el uso de la sentencia *Assign,* este procedimiento realiza la operación de asignar un archivo mediante una correspondencia entre una variable tipo archivo con un archivo externo situado en un disco. La estructura a continuación nos muestra la sintaxis:

Assign (*variable tipo archivo, nombre físico*)

En donde ***variable tipo archivo*** se refiere al nombre interno del archivo por el que se conoce el archivo dentro del programa por ejemplo, el utilizado en la declaración. Y el ***nombre físico*** en este caso se refiere al nombre externo con el que se conoce el archivo por el sistema operativo (por ejemplo: **'c:mi grupo.dat'**); es una cadena que se define como una constante en el programa o bien una variable cuyo valor se lee interactivamente; puede ser una unidad: nombre extensión o bien la ruta completa a un archivo.

Una ves asignado se tiene que abrir el archivo para poder usarlo, para esto se cuenta con varias alternativas:

- ***Rewrite (salida).*** Crea el archivo para escritura, y si ya existe lo sobre escribe.
- ***Reset (salida)***. Este procedimiento abre un archivo existente para una lectura.
- ***Append (salida)***. Abre un archivo existente para añadir datos al final del mismo.

Cuando se termina de usar un programa que maneje archivos, se tiene que cerrar, en este caso se cierra de la misma forma independientemente de la forma que se abrió, esto se logra con ayuda de la sentencia ***close.***

- ***Close (salida).*** Cierra un archivo al terminar su proceso de entrada.

Se muestra un ejemplo en el cual se genera un archivo y se escriben datos

Ejemplo:

```
Var
   nombre: string; salida: textfile;
   edad: integer;
Begin
   Assign (saluda, 'uno.txt');
   write ('¿Cuál es tu nombre?');
   readln (nombre);
   rewrite (salida);
   write (salida, nombre);
   close (salida)=;
End.
```

En este caso el programa anterior genera un archivo externo que no existía anteriormente y lo nombro ***uno.txt***, esto quiere decir que es un archivo de bloc

de notas llamado ***uno***, para seguir escribiendo en el se usa el *append* en lugar de *rewrite.*

El siguiente ejemplo es un programa que muestra datos de un archivo ya existente.

Ejemplo:

```
Var
  nombre: string;
  salida: text;
  edad: integer;
Begin
   Assign (salida, 'uno.txt');
   write ('El nombre es:');
   reset (salida);
   readln (salida, nombre);
   write (nombre);
   close (salida);
End.
```

este programa muestra los datos que se tienen en el archivo que ya existe llamado ***uno.txt*.**

2.6. Funciones

Una función es un módulo de un programa separado del cuerpo principal, que realiza una tarea específica y que puede regresar un valor a la parte principal del programa u otra función o procedimiento que la invoque.

La forma general de una función es:

Function NomFuncion(parámetros:tipo): tipodatoregresa;

Donde *tipodato* especifíca el tipo de dato que regresara la función.

Y el *NomFuncion* tiene dos papeles en pascal:

a) Es el nombre que se invocara dentro del principal o de algún procedimiento u otra función.

b) Es también una variable que deberá cargarse dentro del cuerpo de instrucciones (begin ..end) para que pueda regresar el dato o resultado al principal o procedimiento o función que la este invocando.

La lista de parámetros formales es una lista de variables separadas por punto y coma (;) que almacenaran los valores que reciba la función, estas variables actúan como locales dentro del cuerpo de la función.

Recordar además que cuando se llame una función deberá haber una variable que reciba el valor que regresara la función, es decir, generalmente se llama una función mediante una sentencia de asignación,

Ejemplo:

```
program funciones;
uses crt;
var
   dolar:real;
   FUNCTION dolares(pesos,tc:real):real;
begin
   dolares := pesos /tc;
end;

begin
clrscr;

(* llamando funcion y cargando resultado *)
   dolar := dolares(123.45 , 11.25);
   write('SON ',dolar:0:2, ' DOLARES');
   readln;
end.
```

El resultado que arrojaría este programa de funciones es el siguiente:

SON 10.97 DOLARES

2.7. Recursividad

La recursividad es una técnica que consiste en hacer que, bajo determinadas condiciones, un programa se llame a si mismo.

En el caso del manejo de funciones un ejemplo claro para poder observar la recursividad, es desarrollar un programa que nos muestre el factorial de un número. Las funciones recursivas por lo general se almacenan en pilas.

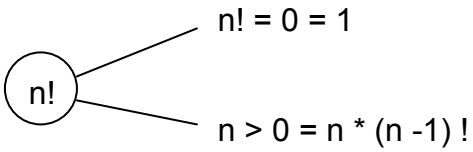
Ejemplo Factorial

5! = 5 * 4 * 3 * 2 * 1 * 1
4! = 4 * 3 * 2 * 1 * 1

Si se observa el procedimiento que se requiere para sacar la factorial de un número, se llegó a la siguiente conclusión:

5! = 5 * 4

Por lo tanto, haciendo un análisis más general del procedimiento, se obtiene lo siguiente:



5! = 5 * 4!
5! = 5 * 4 * 3!
5! = 5 * 4 * 3 * 2!
5! = 5 * 4 * 3 * 2 * 1!
5! = 5 * 4 * 3 * 2 * 1 * 1!
5! = 5 * 4 * 3 * 2 * 1 * 1

Se observó en este ejemplo que es una función que se llama a ella misma pero que siempre termina. A continuación se muestra el código que ayuda a resolver el problema de factorial con el uso de funciones:

```
Funtion factorial (n: longint): longint;
Begin
  If n = 0 then
     factorial:= 1
 else
     factorial:= n * factorial (n – 1);
end;
Var
    valor: longint;
    resultado: longint;
Begin
   writeln (‘Introduce un numero para sacar el factor :’);
   readln (valor);
   resultado:= factorial (valor);
   writeln (‘El resultado es: ‘ , resultado);
   readln;
End.
```

El código anterior es un claro ejemplo del uso de la recursividad de la función para lograr el resultado.

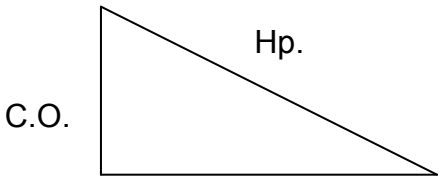
Antes de comenzar a desarrollar el siguiente tema, se vio un ejemplo desarrollado con Pascal usando todas o la mayoría de las sentencias que se vieron a través del capítulo, esto para ver una aplicación general del lenguaje. Desarrollando un programa que tenga una aplicación común.

2.8. Ejemplos y aplicaciones de Pascal

Pascal se puede utilizar para todo tipo de aplicaciones pero su principal finalidad son las aplicaciones científicas. En este caso que se enfoca principalmente a el nivel bachillerato, se tienen muchas aplicaciones para la solución de problemas comunes que se presentan. Realmente el método para

enseñar lenguaje Pascal debe enfocar los ejemplos a la solución de una tarea común de sus alumnos. Por ejemplo, crear un programa que resuelva la factorización de un binomio, o que de el resultado de una derivada o de una integral. Es decir, motivar al estudiante que con ayuda del lenguaje Pascal, puede desarrollar programas que faciliten tareas cotidianas y que finalmente es una herramienta de gran utilidad en todas las áreas en las que se desarrolla.

Programa de un triángulo



C.O. = Hp * Sen (Θ)
C.A. = Hp * Cos (Θ)

Este es un programa que calcúla los catetos de un triángulo-rectángulo

Hipotenusa = 20
Angulo Θ = ?
Calcula C.O. y C.A.

```
Program Triangulo;
Uses crt;
Var
     Hipotenusa:double;
     Angulo:double;
     Cateto_A:double;
     Cateto_O:double;
Begin clrscr;
     Writeln ('Dame el valor de la hipotenusa:');
     Readln (Hipotenusa);
     Writeln ('Dame el valor del ángulo: ');
     Readln (Angulo);
     Cateto_O:= Hipotenusa * Sen(Angulo);
     Writeln (' El Cateto Opuesto es: ');
     Readln (Cateto_O);
     Cateto_A:= Hipotenusa * Cos(Angulo);
     Writeln (' El Cateto Adyacente es: ');
     Readln (Cateto_A);
End
```

Programa que calcula el impuesto del salario

```
Program impuesto;
Var
     Nombre: string;
     Ingreso: integer;
```

```
      IVA: double;
      Isr: double;
      Gastar: integer;
      Impuesto: real;
      Total: real;
Begin
    Writeln ('¿Cuál es tu nombre? ');
    Readln (Nombre);
    Writeln (' ¿Cuál es tu ingreso?');
    Readln (Ingreso);
       If Ingreso < 1500 then
          Isr= 0;
       If Ingreso > 1500 then
         Isr= Ingreso * 0.35;
       Gastar:= Ingreso – Isr;
        IVA:= Gastar * 0.15;
        Impuesto:= Isr + IVA;
    Writeln ('El impuesto que pagó fue :' impuesto);
    Readln (Impuesto);
    Total:= Ingreso – Impuesto;
    Writeln ('El total para gastar es:' Total);
    Readln (Total);
End.
```

Ejemplo de programa "agenda personal" desarrollado con Pascal

Se desarrollo un pequeño programa que maneja una Agenda Personal, la cual se programo en lenguaje Pascal, esta agenda cuenta con varias opciones, como se muestra en la Figura 2.5.

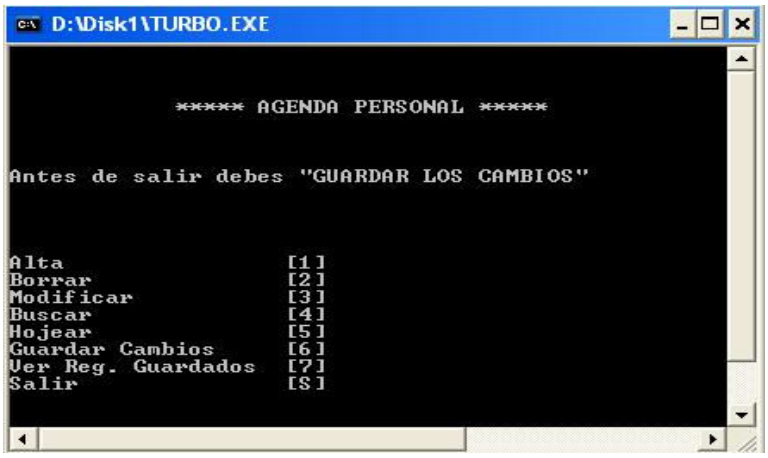


Figura 2.5. Agenda personal

Se puede observar que cuanta con las opciones de:

1. Alta
2. Borrar
3. Modificar
4. Buscar

5. Hojear
6. Guardar cambios
7. Ver registros guardados
8. salir

Cada una de estas opciones fueron programadas en el lenguaje Pascal, a continuación se muestra el código que se utilizó para la realización de esta agenda con el fin de analizarlo y conocer como fue el manejo de las sentencias, y el manejo de archivos para lograr su funcionamiento.

"Programa AGENDA "

```
USES
  crt;
TYPE
  AGENDA=record   (*Este registro contienes las variables de la Agenda*)
    nombre,correo_e,domicilio,comentario,tel:string;
  end;
CONST
  t_agenda=5;    (*Esta variable cambia el tamaño del arreglo en todo el programa*)
VAR
 opcion:char;
  REGISTRO:array [1..t_agenda] of AGENDA;
  archivo: file of AGENDA;
  num,cont:integer;

PROCEDURE agregar;
 begin
   clrscr;
   writeln('Numero de alta a ingresar:');
   readln(num);

   writeln('Nombre:');
   readln(registro[num].nombre);

   writeln('Telefono:');
   readln(registro[num].tel);

   writeln('Correo electr¢nico:');
   readln(registro[num].correo_e);

   writeln('Domicilio:');
   readln(registro[num].domicilio);

   writeln('Comentario:');
   readln(registro[num].comentario);
   writeln('El registro quedo almacenado!!!');
   writeln;
   writeln('ENTER para regresar al menu anterior');
   readln;
 end;

PROCEDURE hojear;
begin
 clrscr;
 for cont:=1 to t_agenda do
```

```
  begin
    writeln(registro[cont].nombre);
    writeln(registro[cont].tel);
    writeln(registro[cont].correo_e);
    writeln(registro[cont].domicilio);
    writeln(registro[cont].comentario);
    writeln;
    readln;
  end;
   write('ENTER para regresar al menu anterior');
   readln;
end;

PROCEDURE modificar;
begin
  clrscr;
  writeln('Numero de registro a modificar:');
  readln(num);

  writeln('Nombre:');
  readln(registro[num].nombre);

  writeln('Telefono:');
  readln(registro[num].tel);

  writeln('Correo electronico:');
  readln(registro[num].correo_e);

  writeln('Domicilio:');
  readln(registro[num].domicilio);

  writeln('Comentario:');
  readln(registro[num].comentario);

  writeln('El registro ha sido modificado');
  writeln;
  writeln;
  writeln('ENTER para regresar al menú anterior');
  readln;
end;

PROCEDURE borrar;
begin
  clrscr;
  writeln('Que número de alta deseas borrar: ');
  readln(num);
  registro[num].nombre:=' ';
  registro[num].tel:=' ';
  registro[num].correo_e:=' ';
  registro[num].domicilio:=' ';
  registro[num].comentario:=' ';
  writeln('El registro esta borrado!!!');
  writeln;
  writeln;
  writeln('ENTER para regresar al menú anterior');
  readln;
end;
```

```
PROCEDURE salvar_a_disco; (*Permite salvar al Disco Duro el arreglo AGENDA*)
 var
   temp:agenda;
   cont:integer;
 begin
   clrscr;
   rewrite(archivo);

     for cont:=1 to t_agenda do
       begin
         temp:=registro[cont];
         write(archivo, temp);
       end;

   close (archivo);
   writeln;
   writeln('Los registros fueron guardados satisfactoriamente!!!');
   writeln;
   writeln;
   writeln('ENTER para regresar al menú anterior');
   readln;
 end;

PROCEDURE leer_de_disco;
var
   cont:integer;
   temp:agenda;
begin
   clrscr;
   reset(archivo);  (*Abre el archivo de solo lectura*)
   for cont:=1 to t_agenda do
      begin
        read(archivo, temp);
        registro[cont]:=temp;
      end;
        close (archivo);
        writeln('Los registros son:');
        writeln;
        writeln;
       hojear;
     end;

PROCEDURE consultar;
 var
  num_lista:integer;
 begin
  clrscr;
  writeln('Numero de Registro a consultar: ');
  readln(num);
  writeln('Su nombre es: ', registro[num].nombre);
  writeln('Su telefono es: ', registro[num].tel);
  writeln('Su correo electrónico es: ', registro[num].correo_e);
  writeln('Su domicilio es: ', registro[num].domicilio);
  writeln('Comentario: ', registro[num].comentario);
  writeln;
  writeln;
  writeln('Escribe ENTER para regresar al menú anterior');
```

```
  readln;
  end
BEGIN
  opcion:=' ';
  assign (archivo, 'agenda.txt');
  while opcion<> 'S' do
  begin
  clrscr;
  writeln;
  writeln;
  writeln;
  writeln('          ***** AGENDA PERSONAL *****     ');
  writeln;
  writeln;
  writeln;
  writeln('Antes de salir debes "GUARDAR LOS CAMBIOS"');
  writeln;
  writeln;
  writeln;
  writeln;
  writeln ('Alta             [1]');
  writeln ('Borrar           [2]');
  writeln ('Modificar        [3]');
  writeln ('Buscar           [4]');
  writeln ('Hojear           [5]');
  writeln ('Guardar Cambios     [6]');
  writeln ('Ver Reg. Guardados  [7]');
  writeln ('Salir             [S]');
  writeln;
  readln (opcion);
  opcion:=upcase(opcion);
    case opcion of
     '1':agregar;
     '2':borrar;
     '3':Modificar;
     '4':consultar;
     '5':Hojear;
     '6':Salvar_a_disco;
     '7':Leer_de_disco;
    end;
  end;
end.
```

Analizando el código anterior, se observó que maneja la captura de los datos por medio de un archivo llamado "*agenda.txt*" el cual es manipulado por medio de código, en este caso, el programa logra sacar los datos para mostrarlos por medio de la opción Buscar el cual muestra los datos almacenados en el archivo en la pantalla. (Figura 2.6)

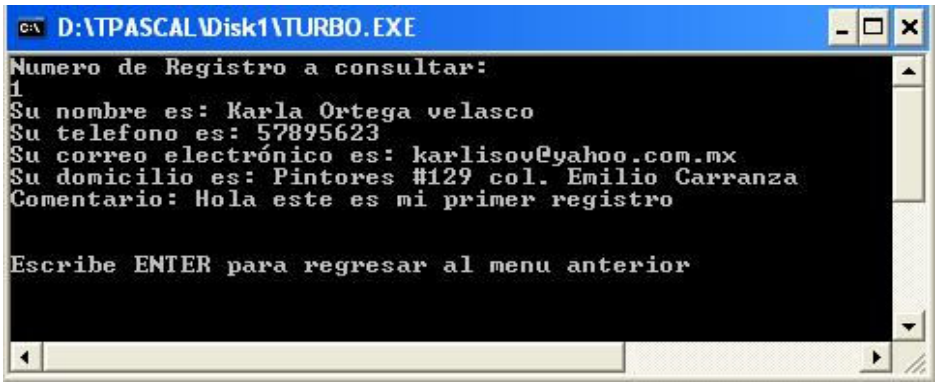


Figura 2.6. Registro de Agenda Personal.

Con ayuda del ejemplo anterior es mas sencillo entender el por que es necesario aprender a programar un lenguaje como Pascal, ayuda a resolver problemas y cubre las necesidades que se presentan.

Hasta aquí, se ha logrado desarrollar algunos de los conceptos básicos del lenguaje y las diferentes características con las que cuenta, se observo que es un lenguaje sencillo de manejar, esto con ayuda de algunos ejemplos que se manejaron de las diferentes sentencias que usa este lenguaje, así que por otro lado se enfocó en el uso de las diferentes herramientas con las que cuenta el lenguaje de programación Delphi para lograr hacer programas mas sencillos para los usuarios, es decir, de manera mas gráfica, sin dejar a tras el uso del lenguaje Pascal que se vio anteriormente.

2.9. Lenguaje de programación Delphi

Comúnmente, cuando se desarrolla un programa en un lenguaje de programación para MS-DOS como lo es Pascal, se empieza por construir este con los procedimientos y funciones que proporcionan las librerías del lenguaje de programación. Esto ocurre principalmente cuando se empieza con programas sencillos.

Conforme va surgiendo la necesidad de mejorar la calidad de los programas, se hace necesario crear librerías personalizadas que contengan funciones para cálculos, conversiones numéricas específicas, captura y validación de datos, pantallas de presentación y en un nivel más avanzado algoritmos para desplegar gráficas de mapas de bits en diversos formatos e iconos así como menús desplegables, ventanas y botones para dar una presentación mucho mas profesional a los programas.

Todo el proceso para generar librerías personalizadas requiere de mucho esfuerzo. En algunos casos, ese esfuerzo es en vano debido a que:

- Las librerías personalizadas rara vez se documentan.

- La mayoría de las veces son programadas en una forma procedimental, por lo que requieren de varias variables globales y constantes para su operación, lo cual ocasiona que posteriormente se confundan unas con otras y se torne difícil manejarlas dentro de los programas en los que se incluyan. Esto es algo que no sucede con la programación orientada a objetos, debido a que se encapsulan los datos con el código de tal forma que las variables requeridas pertenecen a los objetos y de esta manera es menos probable confundir a cual objeto pertenecen.
- Entre más completa es una librería personalizada, más complejo se torna aplicarla a los programas.
- Al principio es fácil utilizar el código de la librería personalizada en los programas; pero, conforme pasa el tiempo y se olvida como fue programada, se torna más difícil su uso.
- Los elementos de la librería personalizada rara vez guardan similitud entre ellos, por lo que resulta difícil combinarlos. Para esto se debe estudiar minuciosamente como funciona cada uno de ellos.

Analizando todo lo anterior sobre las librerías personalizadas y recordando que un hay dilema que se discute comúnmente entre los programadores que dice: "Para que programar algo si ya esta hecho". Algunos votan por que se debe empezar desde cero para aprender a desarrollar cualquier cosa; otros opinan que es mejor tomar lo que ya fue programado, probado y depurado por profesionales. De esta forma se puede avanzar más rápido en el largo camino para llegar a ser un programador más eficiente en el desarrollo de aplicaciones de software de propósito específico.

Sin duda, los lenguajes de programación visual resuelven casi todos los problemas que involucra el utilizar complejas librerías personalizadas. Los lenguajes de programación visual se diseñaron con la filosofía de que un lenguaje debe proporcionar todos los elementos necesarios y estos elementos deben ser fáciles de utilizar por cualquier programador, aunque no sea un experto.

La solución al problema de las librerías personalizadas es clara, pero comúnmente los programadores piensan que es más difícil aprender un lenguaje de programación visual desde cero que empezar a programar sus propias librerías. Aunque se ha observado que es mas fácil aprender un lenguaje de programación visual que tratar de igualar todas sus ventajas con librerías personalizadas.

Cambiar de un lenguaje de programación para MS-DOS a un lenguaje de programación visual resulta sencillo. Por ejemplo, la enorme diferencia que hay entre Pascal y Delphi es que con este último podemos desarrollar un producto de software de gran calidad y con Pascal tan sólo se puede aprender a programar e intentar desarrollar un producto de software de gran calidad.

Delphi es una potente herramienta de desarrollo de programas que permite la creación de aplicaciones para Windows 3.x, Windows95, Windows NT y

Windows XP. Esto quiere decir que Delphi esta basado en un tipo de programación visual que se le conoce como programación orientada a eventos.

2.9.1. Programación orientada a eventos

Los lenguajes visuales orientados a eventos y con el manejo de componentes dan al usuario que no cuenta con mucha experiencia en desarrollo, la posibilidad de construir sus propias aplicaciones utilizando interfaces gráficas sobre la base de ocurrencia de eventos.

Para soportar este tipo de desarrollo interactúan dos tipos de herramientas, una que permite realizar diseños gráficos y , un lenguaje de alto nivel que permite codificar los eventos. Con dichas herramientas es posible desarrollar cualquier tipo de aplicaciones basadas en el entorno.

Delphi es uno de los lenguajes de programación que más entusiasmo despiertan entre los programadores de computadoras, tanto expertos como novatos. En el caso de los programadores expertos por la facilidad con la que desarrollan aplicaciones complejas en poquísimo tiempo comparado con lo que cuesta programar por ejemplo en lenguaje C++. En el caso de los programadores novatos por el hecho de ver de lo que son capaces a los pocos minutos de empezar su aprendizaje.

Es un lenguaje de programación visual, también llamado lenguaje de 4° generación. Esto quiere decir que un gran número de tareas se realizan sin escribir código, simplemente con operaciones gráficas realizadas con el ratón sobre la pantalla, a esto se le llama eventos.

Ya se ha dicho que las acciones del usuario sobre el programa se llaman ***eventos***. Son eventos típicos el clic sobre un botón, el hacer doble clic sobre el nombre de un fichero para abrirlo, el arrastrar un icono, el pulsar una tecla o combinación de teclas, el elegir una opción de un menú, el escribir en una caja de texto, o simplemente mover el ratón. Más adelante se vieron los distintos tipos de eventos reconocidos por Delphi. Cada vez que se produce un evento sobre un determinado tipo de control, Delphi arranca una determinada función o procedimiento que realiza la acción programada por el usuario para ese evento concreto. Estos procedimientos se llaman con un nombre que se forma a partir del nombre del objeto y el nombre del evento.

La programación en Delphi es un entorno de desarrollo flexible y potente. Además es intérprete de un lenguaje llamado Object Pascal.
Programación Delphi no es solo un intérprete, sino que además incluye otras herramientas para facilitar la escritura del código y el diseño de la aplicación
Una de las mayores ventajas de la programación delphi es que es una programación orientada a objeto.
Se conoce como Objeto a los diferentes componentes visuales con que trabaja Delphi. Con lo que son objetos una ficha, un botón, una lista, etc..

La ventana principal de programación Delphi se denomina Paleta de Componentes.

Un programa en Delphi consta de las siguientes secciones:
Un compilador: el cual crea el ejecutable
Una librería: conocida como VCL, la cual es una librería de clases.

Iniciemos por conocer el entorno de trabajo que nos ofrece Delphi.

2.9.2. IDE de Delphi

El IDE viene siendo el ambiente de desarrollo integrado (integrated development environment) o el entorno de Delphi (Figura 2.7)

El primer paso que se dio para poder trabajar con Delphi fue conocer su entorno, donde existen múltiples ventanas, conteniendo opciones, botones, propiedades, etc. Por lo tanto se dio un recorrido para conocer un poco de los elementos más importantes.

Al ejecutar Delphi, en pantallas por lo regular siempre se encuentra con los mismos elementos. Son precisamente los que se utilizaron con mayor frecuencia.

Lo siguiente esta enfocado para entender como funciona una aplicación en el ambiente Delphi dentro de Windows, y cuales son las principales partes de un programa de "Object Pascal" en Windows (object pascal es el nombre formal del lenguaje de programación, así como en Visual Basic el lenguaje es Basic, en Delphi el lenguaje es Object Pascal).Cuando inicia el programa por primera vez, se le presenta un formulario en blanco y el IDE como se muestra en la figura siguiente:

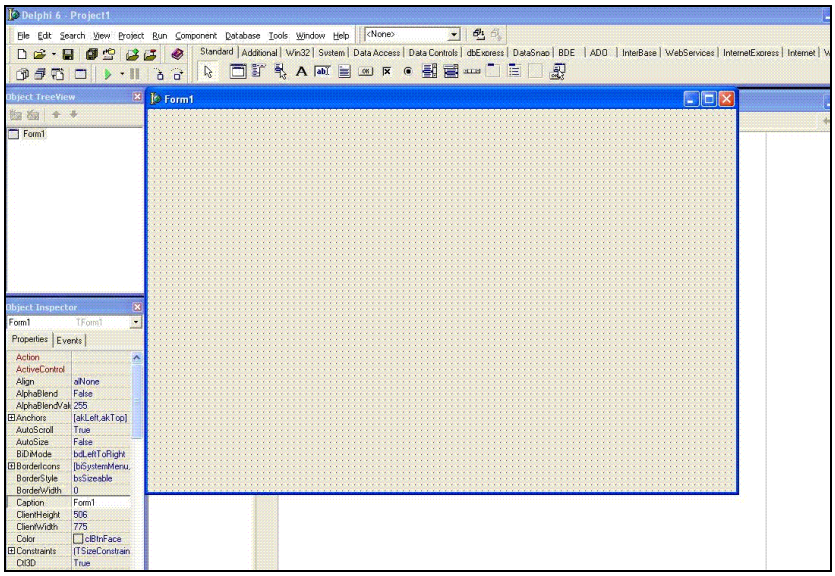


Figura 2.7. IDE de Delphi.

El IDE de Delphi se divide en tres partes. La ventana superior puede considerarse como la principal. Contiene las barras de herramientas y la paleta de componentes. La barra de herramientas de Delphi le ofrece acceso con un solo clic a tareas como abrir, guardar y compilar proyectos. La paleta de componentes contiene un amplio conjunto de componentes que puede colocar sobre su formulario, como pueden ser etiquetas de texto, controles de edición, cuadros de listas, botones, etc. Por comodidad los componentes están divididos por grupos.

Para poder colocar un componente en un formulario, simplemente se da clic en cualquiera de las opciones que se encuentran en la barra de herramientas y después se da un clic a el formulario en donde se desea que aparezca el componente y listo, aun no es necesario conocer el funcionamiento de los componentes.

Un componente es una pieza binaria de software independiente, que realiza cierta función específica predefinida, como una etiqueta de texto, un control de edición o un cuadro de lista.

El object inspector

En la Figura 2.8 se muestra el Object Inspector, el cual ayuda a modificar las propiedades y eventos de un componente, constantemente se trabajo con esta herramienta a la hora de diseñar un programa. El Object Inspector tiene dos fichas: *Propierties* y *Events.* Las **propiedades** de un componente controlan la forma en que este opera. Por ejemplo, cambiar la propiedad de *Color* de un componente modificar el color de fondo de ese componente. La lista de propiedades disponibles varia entre un componente y otro, aunque por lo regular los componentes tiene varios elementos en común. Una propiedad determina la operación de un componente.

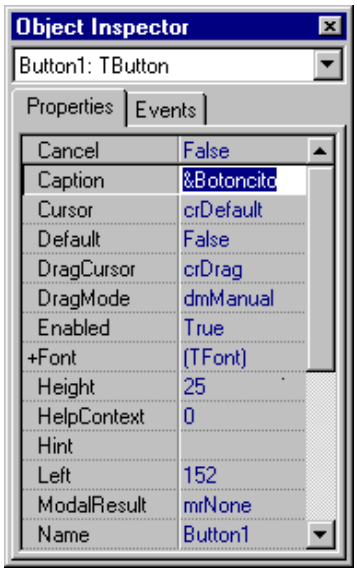


Figura 2.8. Object Inspector.

La ficha Events contiene una lista de eventos de un componente, los eventos ocurren al interactuar con el componente. Por ejemplo, al hacer clic en un componente, se genera un evento que le indica que se hizo un clic en él. Puede escribir código que responda a estos eventos, realizando acciones específicas cuando ocurre un evento. Al igual que las propiedades, los eventos a los que puede responder varían de un componente a otro.

- un *evento* es algo que ocurre como resultado de la interacción de un componente con el usuario o con Windows. Y un controlador de eventos es una sección de código que se invoca en su aplicación en respuesta a un evento.

El espacio de trabajo de Delphi. La parte principal del IDE de Delphi es el espacio de trabajo. Inicialmente, el espacio de trabajo despliega el **Form Designer**. En Delphi, un *formulario* representa una ventana en su programa, este podría ser la ventana principal del programa que este diseñando o bien, un cuadro de dialogo o cualquier otro tipo de ventana, el formulario se emplea para colocar, mover y dimensionar componentes como parte del proceso de creación del formulario.

El Code Editor está oculto tras el Form Designer. Aquí es donde se introduce el código al escribir sus programas. El Object Inspector, Form Designer, Code Editor y la paleta Component trabajan de manera interactiva mientras usted construye sus aplicaciones.

Ya que se vio que es lo que conforma el IDE de Delphi, hagamos algunos ejemplos para poder comprender mejor su funcionamiento.

2.9.3. Programa“Hola Mundo” con Delphi

Durante el curso se desarrollo el programa “Hola Mundo” como es tradicional en todos los lenguajes de programación, donde se utilizaron las herramientas de Delphi.

Inicialmente se debe tener operando Delphi para obtener un formulario en blanco. De esta manera predeterminada, el formulario se llama *Form 1*. a la izquierda del formulario, el Object Inspector muestra las propiedades del mismo, en este caso en la propiedad *Caption* se escribe “Hola Mundo” para modificar el título del formulario. Ahora se logra ver que el formulario se llama Hola Mundo efectivamente, pero se necesita hacer más para lograr desplegar un “Hola Mundo” en el formulario, así que se, necesita agregar el texto en el centro del formulario, para lograrlo se debe agregar una *etiqueta de texto*. Observemos como se puede lograr:

1. se da clic en la barra de componentes a uno de ellos que muestra una A, si se pasa el mouse se despliega una información que dice *Label.*

2. en cualquier parte del formulario se da un clic. Un componente de etiqueta se colocará sobre el formulario con el titulo predeterminado *Label 1.*
3. a continuación, en el Object Inspector, en este momento despliega las propiedades de *Label 1*, estará en la propiedad *Caption.*
4. en *Caption* se escribe “Hola Mundo” y en ese momento muestra como se visualizan estas palabras sobre el formulario.
5. mientras se trabaja con esta etiqueta, también se puede cambiar el tamaño de la etiqueta, o cambiar el tipo de letra con la propiedad Font, esta muestra un menú con las diferentes fuentes con las que se cuenta.
6. también se puede modificar el tamaño de la fuente con la propiedad Size, así se logra instantáneamente el cambio en el formulario.
7. finalmente se oprime el menú Run y o en su defecto F9 y se lograra ver la corrida del programa. Se observa el mismo formulario pero ahora con los cambios establecidos.

Probablemente la etiqueta que se colocó en el formulario no esta centrada y tal vez desee moverlo. Para mover un componente simplemente haga clic en él y arrástrelo a la posición adecuada, ahora puede recompilar el programa y correrlo.

Con este ejemplo, se observa que es mas interesante escribir programas en Windows con Delphi que en los viejos tiempos.

Para terminar con el programa es necesario guardarlo, así que, el programa preguntará si se desea guardar como proyecto 1, en este caso se puede poner el nombre que mas se convenga, pero es necesario explicar que al trabajar con el entorno de Delphi se deben guardar varios archivos, el proyecto y el formulario, más adelante se vera a que se refiere con esto.

Se inicia explicando como trabaja Delphi y su entorno visual.

Delphi cuenta con un compilador/encadenador el cual es un programa que crea el archivo ejecutable de Windows (PE) estilo Intel, sin ningún interprete de por medio. La librería es código que 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.

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 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 sólo funciona en Delphi.

Archivo de proyecto en Delphi DPR

La Figura 2.9 muestra el archivo DPR que se crea cuando comienza Delphi. En general 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 2.9. Archivo DPR.

Si se analiza este archivo. La sentencia "*program project1*" dice que el programa se llama *project1*. El nombre de programa debe ser el mismo que el nombre del archivo (sin extensión).

La sección *USES* muestra que archivos son usados. *Forms* es la librería de Delphi que contiene los comandos para crear y manipular ventanas (que en Delphi se llaman Formas). El otro renglón en la sección "uses" dice que la unidad llamada *"Unit1"* se encuentra en el archivo *"unit1.pas",* y Delphi nos hace un comentario *{ }* que dice que la forma en esta unidad se llama *Form1.*

*$R* es una "directiva del compilador". Una directiva es un comando para el compilador, no para el IDE. Cuando el compilador encuentra la directiva *{$R *.RES}* un archivo de *"recursos"* con el mismo nombre del programa (pero con extensión RES) es encadenado junto con el programa.

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" (aunque podría ser invisible) hasta que el Application Run termina. Después de esto (normalmente cuando "Run" recibe el mensaje "Termínate") el programa termina.

RUN en Delphi es un método de la aplicación que hace al programa entrar en el ciclo de mensajes de Windows. El ciclo de mensajes es un ciclo estilo "Do

While" que recibe mensajes de Windows (clic de botones, tecleos, movimientos del mouse) y procesa los mensajes en una manera consistente para todas las "ventanas" de la aplicación.

Con el ejemplo que se hizo con el programa de “Hola Mundo” se logró trabajar con todas estas funciones que ya contiene Delphi para ayudar a que todo sea más sencillo y que todo se desarrolle automáticamente para ya no preocuparse por todas las librerías que se necesitan, ya que todas estas ya están preconstruidas.

Se desarrollo el mismo programa pero utilizando más herramientas como un botón, es decir la meta de este ejercicio consiste en hacer que aparezca sobre la pantalla las palabras “Hola Mundo” al oprimir un botón, para este caso debemos de tomar en cuenta otras aplicaciones, vamos a comenzar siguiendo los pasos que a continuación se presentan:

1. Debe estar nuevamente en el entorno de Delphi para comenzar con un nuevo proyecto.
2. De clic en la ficha Standard de la paleta Component y otro en el icono que tiene un **OK** sobre él (componente de *Button*).
3. Coloque su puntero en cualquier parte del formulario y haga clic. En el formulario aparecerá un botón.
4. Elija un componente Label y colóquelo cerca del centro del boton, esto para darle que aparezca un nombre sobre el botón.

Se puede observar que los componentes que se colocaron en el formulario tienen el título predeterminado *Label 1* y el botón tiene *Button 1.* en este ejercicio modificará el título de la etiqueta a través del código. Al cambiar las propiedades de un componente a través del Object Inspector y el Form Designer, se dice que esta realizando un cambio en *tiempo de diseño*, mientras que al modificar una propiedad a través de código que se invoca al ejecutar el programa, se dice que esta efectuando un cambio en *tiempo de ejecución*.

Para cambiar la propiedad *Caption* en tiempo de ejecución, se siguen los siguientes pasos:

1. Haga doble clic en el botón que esta sobre el formulario. Tan pronto como lo hace, Delphi genera un controlador de evento para el evento OnClick del botón. El código generado luce como el siguiente:

   ```
   Procedure TForm1.Button1Click(Sender : TObject);
   Begin
   End;
   ```

2. En este momento sólo se necesita entender que el controlador del evento OnClick es una sección de código que se ejecutará cada vez que se haga clic en el botón (es decir, siempre que el programa esta en ejecución). El cursor de edición se coloca entre las instrucciones Begin y

end y está en espera de que se introduzca código. Se escribe el siguiente código en donde se encuentra el cursor:
Label1.Caption := 'Hola Mundo';

Ahora el controlador de eventos completo luce como sigue:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Caption:='HOLA MUNDO';
end;
end.
```

Este código es muy sencillo y simplemente lo que hace es asigna el valor de Hola Mundo a la propiedad *Caption* de la etiqueta. Para poder observar el resultado se hace la ejecución del programa antes y después de darle clic en el botón como se muestra en las Figuras 2.10.

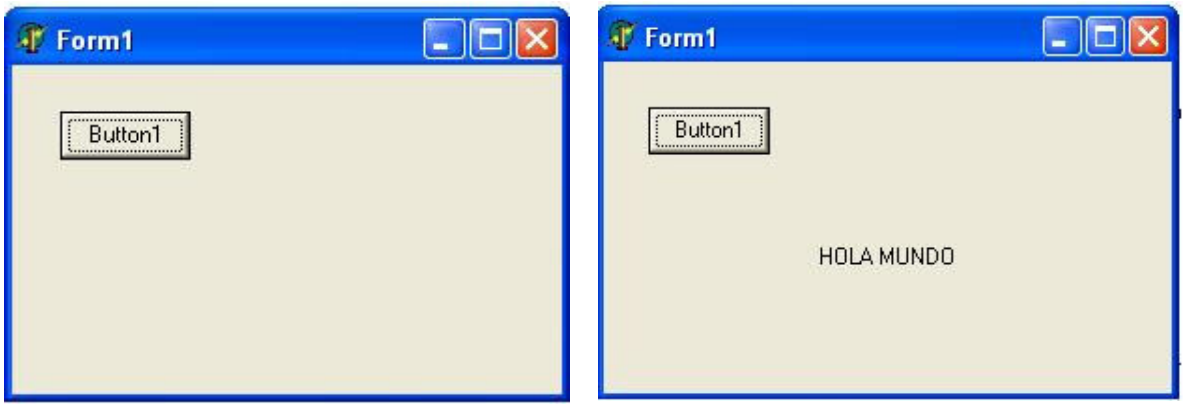


Figura 2.10. Ejecución del Programa "Hola Mundo" antes y después.

Delphi cuenta con una gran variedad de herramientas que son de uso muy común, como lo son las etiquetas de texto, el cuadro de texto, los botones entre otros, en este módulo solo se vieron algunas características que se consideran importantes para el uso de estas herramientas, para más adelante poder realizar un ejercicio usando algunas de estas.

2.10 Principales herramientas de Delphi

En la paleta de componentes *Standard* se cuenta con varios controles o herramientas. Mediante estos controles se puede manipular eventos que requieren nuestros programas, a continuación se mostraran los mas indispensables para desarrollar algún programa en Delphi.

Etiquetas de texto o Label 

Anteriormente se utilizaron las etiquetas de texto o *Label*, lo cual se observó 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 sólo 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 etiquta.

El control Edit 

Uno de los componentes que permite la entrada de datos por teclado es el el control *Edit*. El nombre del control se fija 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 sera fijado mediante la propiedad Color, mientras que los atributos del texto se establecen mediante la propiedad *Font.* Tambien es posible fijar una determinada figura para el cursor del raton 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 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, 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 aparece 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 CheckBox 

Mediante los controles de *CheckBox*, o cajas de selección. Se puede 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 se asigne 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 se puede conocer mediante la propiedad *Checked*, que tomará el valor de Trae si está marcado o el valor *False* en caso contrario.

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.

2.11. Ejemplos y aplicaciones de Delphi

Delphi 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.

Programa "Agenda Personal" desarrollado con Delphi

Se desarrolló un ejemplo muy parecido al que se hizo con el Lenguaje de Programación Pascal una "Agenda Personal", para lograr ver la diferencia en cuanto a uso de sentencias en Pascal y Delphi, también para mostrar como se ve en el programa usando las herramientas visuales con las que cuenta Delphi. Inicialmente se muestra la vista del formulario de la Agenda Personal y posteriormente el código desarrollado para lograr analizar la sintaxis en Delphi, la cual es muy parecida a la sintaxis que se observa en Pascal ya que como anteriormente se explicó que Delphi trabaja con Object Pascal lo cual el lenguaje base de programación que usa Delphi.

Se observa en la Figura 2.11 la corrida de Agenda Personal hecha en Delphi

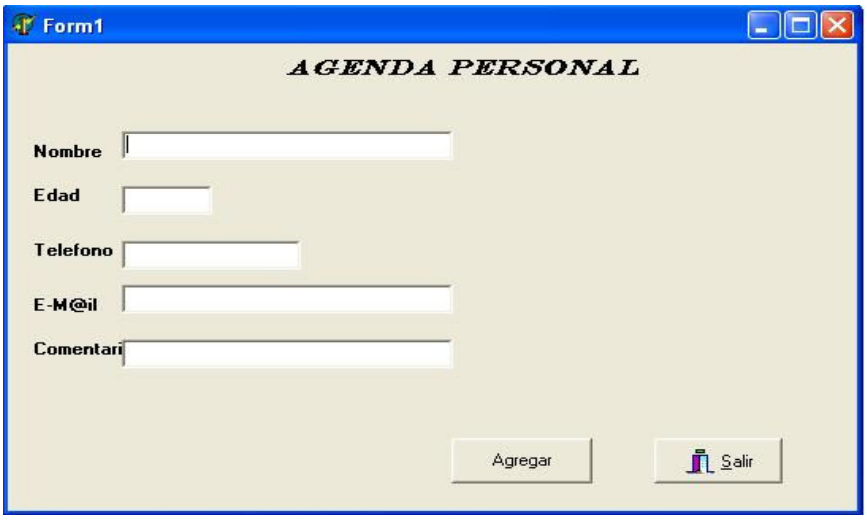


Figura 2.11. Agenda Personal en Delphi.

En este ejemplo únicamente da la opción de agregar elementos a la agenda, y lo almacena dentro de un archivo de texto llamado *"agenda2.txt"* el cual se muestra a continuación en la Figura 2.12.

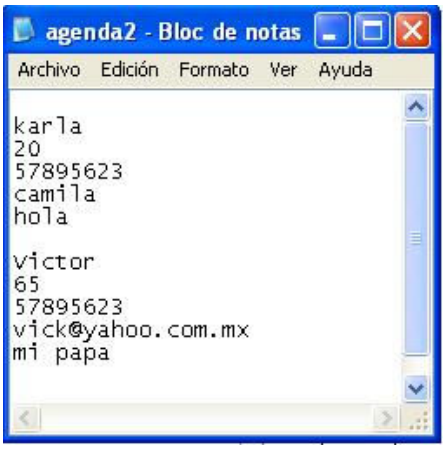


Figura. 2.12. Archivo agenda2

Aquí se observa que cada ves que se agregan datos de un elemento de la agenda lo manda al archivo que se asigno desde el código y aquí se puede ver los datos necesarios. Esto también se realizo en el ejemplo con pascal como se mostró anteriormente.

A continuación se muestra el código principal de la agenda personal en Delphi para poder analizarlo y conocer más de la sintaxis usada en Delphi.

```
unit otraagenda

interface


uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Button1: TButton;
    BitBtn1: TBitBtn;
    Label6: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  archivo:textfile;

implementation
{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
  assignfile(archivo,'agenda2.txt');
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  nombre,edad,telefono,correo,comentario:string;
begin
  nombre:=edit1.Text;
  edad:=edit2.Text;
```

```
  telefono:=edit3.Text;
  correo:=edit4.Text;
  comentario:=edit5.Text;
  append (archivo);
    writeln(archivo,nombre);
    writeln(archivo,edad);
    writeln(archivo,telefono);
    writeln(archivo,correo);
    writeln(archivo,comentario);
    writeln(archivo);
  closefile(archivo);
end;

end.
```

Por medio del ejemplo anterior se muestra la facilidad de poder desarrollar programas con Delphi, se logra dar un aspecto gráfico más sencillo para el manejo de los usuarios finales y el código es igual de sencillo como lo es Pascal que se estudió desde el inicio de este capítulo.

Ya que durante la última década el campo de la computación ha tenido diversas aplicaciones en todas las áreas de conocimiento, y en particular en el campo educativo. En algunos centros educativos a nivel bachillerato se ha comenzado a enseñar a programar inicialmente con programación Pascal, por considerarse un lenguaje de programación sencillo, ya que con pocas instrucciones se puede elaborar programas muy complejos, así se podrán elaborar programas relacionándolos con sus propias tareas escolares. Por lo tanto, para poder lograr buenos resultados se debe estar capacitado con conocimientos básicos del lenguaje, en este caso Pascal y Delphi, por eso el objetivo de este módulo es una actualización de personal docente en programación básica para poder brindar un mejor nivel de educación.

Como se puede observar el desarrollo de los aspectos básicos del lenguaje Pascal son muy amplios, pero también son muy extensas las aplicaciones que podemos desarrollar con ayuda de Delphi, así que lo que se mostró anteriormente son los aspectos básicos para poder desarrollar programas sencillos, por ejemplo inicialmente con pascal fue necesario conocer cada una de las sentencias que se utilizan para desarrollar un programa que nos ayude a resolver un problema especifico de codificación de algoritmos, también es necesario saber la sintaxis correcta de cada una de estas sentencias antes mencionadas, sus limitantes, sus procedimientos de solución, su uso en particular, sin en cambio en la programación con Delphi sólo se explicaron algunas herramientas básicas las cuales nos ayudaron a desarrollar un programa sencillo, pero en realidad nos basamos en lo que aprendimos con pascal ya que la sintaxis y el uso de sentencias son muy parecidas. Sin dejar de tomar en cuenta que la aplicación de este tipo de programación como Delphi es muy extensa y nos ofrece mucho más herramientas como manejo de bases de datos entre otras. Por lo tanto solo dimos un pequeño recorrido a Delphi pero para lograr ver algunas diferencias entre el lenguaje pascal basado en programación con MS-DOS y una programación visual como lo que es Delphi basada en aplicaciones Windows e interfases de usuario.

Otros aspectos que debemos de tomar en cuenta para una buena práctica de la programación es lo siguiente:

Utilizando la programación modular, dividimos el problema en partes más pequeñas, es conveniente que el número de partes sean las suficientes para ser manejables.

Asignar nombres significativos a las variables, es decir, hacer más comprensible el programa.
Es necesario el uso de comentarios durante la etapa de depuración (* hasta aquí esta correcto *), pero es necesario borrarlos al terminar la depuración, se utilizan también para la documentación del código fuente.

En cuanto a la programación con herramientas Delphi, es importante señalar que tienes sus ventajas y desventajas así que debemos saber escoger el lenguaje que más nos convenga según el problema que tenemos que resolver.

CAPÍTULO 3

PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

3.1. Antecedentes

El lenguaje de programación Java fue desarrollado por SUN con la intención de competir con Microsoft en el mercado de la red.[9]

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.

La portabilidad se consigue haciendo 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.

9. García de Jalón, Javier. *Aprenda Java como si estuviera en primero*. p. 9-25.

3.2. Programación orientada a objetos con Java

3.2.1. Características de Java

Las características principales que nos ofrece Java respecto a cualquier otro lenguaje de programación, son[10]:

- ***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 librería nuevas o actualizadas no paralizarán las aplicaciones actuales (siempre que mantengan el API anterior). La figura 3.1 muestra como trabaja el navegador de Java.

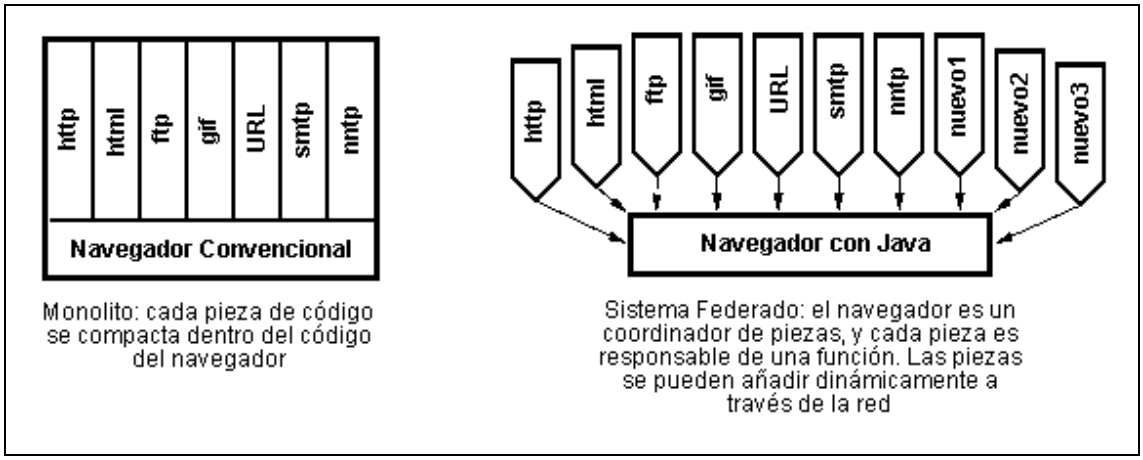


Figura 3.1. Navegador de Java.

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, se vio un poco a cerca de los conceptos de la programación orientada a objetos.

3.2.2. Programación orientada a objetos

Dado que Java es un lenguaje orientado a objetos, es imprescindible entender qué es esto y en qué afecta a los 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, ejecuta 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.

Pero antes de continuar, conozcamos algunos conceptos básicos sobre este tipo de programación para poder comprender mejor su funcionamiento.

3.2.3. ¿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.

3.2.4. ¿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.

3.2.5. ¿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.

3.2.6. 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.

3.2.7. 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.*

10.ibíd.

Los tipos ***primitivos*** contienen un sólo valor e incluyen los tipos como los enteros, coma flotante, los caracteres, etc... En la Tabla 3.1 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 3.1. 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 un 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.

3.2.8. 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 3.2 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 3.2. 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 3.3)

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

Tabla 3.3. 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 3.4).

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 3.4. Operadores de Incremento y Decremento.

3.3 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 3.5)

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

Tabla 3.5. Sentencias de Control de Flujo en Java.

3.3.1. 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 puntuacion;
String nota;

if (puntuacion >= 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.

3.3.2. 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;
  }
```

3.3.3. 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 sóla 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 leido un el carácter = " + contador);
}
```

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

3.3.4. 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 iteracción 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 vió al principio, excepto en que la expresión se evalua 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);
```

3.4. 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.

Como 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***.

Como 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.

Como 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.

Como 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;

Como 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 rectangulo.

```
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().*

3.5. Declaración de 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.

3.6. 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).

3.7. 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.

3.7.1. 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()*.

3.7.2. 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 3.2:

- 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.

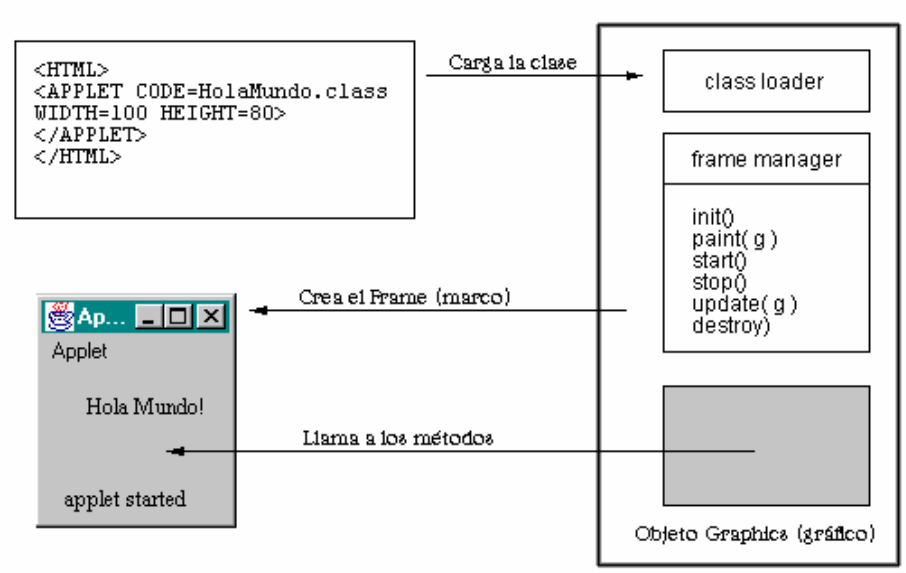


Figura 3.2. Ciclo de vida de un Applet.

3.8. Software para la ejecución de programas Java

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.

3.8.1. 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 3.3

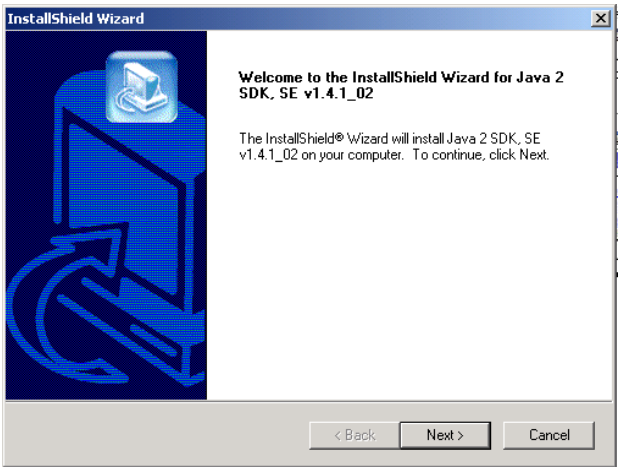


Figura 3.3. Asistente de Configuración SDK

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 3.4).

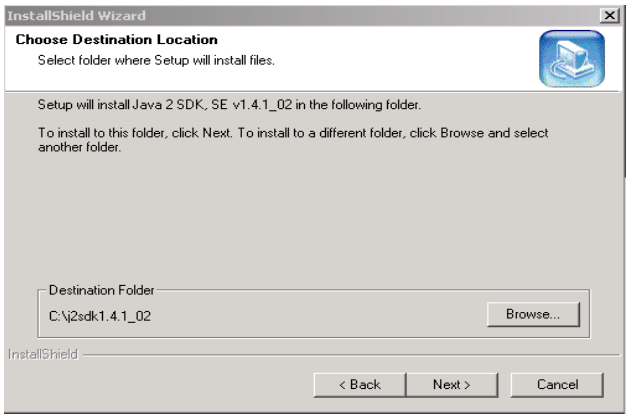


Figura 3.4. Selección de Unidad destino de instalación

3.8.2. 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 3.5 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:

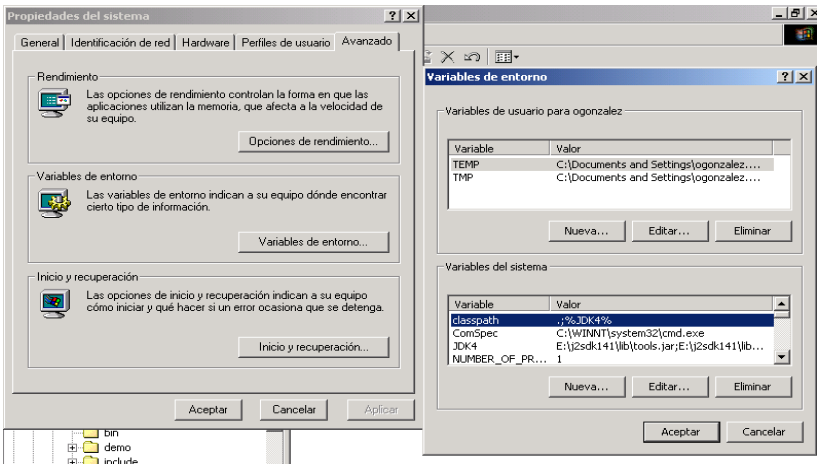


Figura 3.5. Variables de Entorno.

3.8.3. 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:

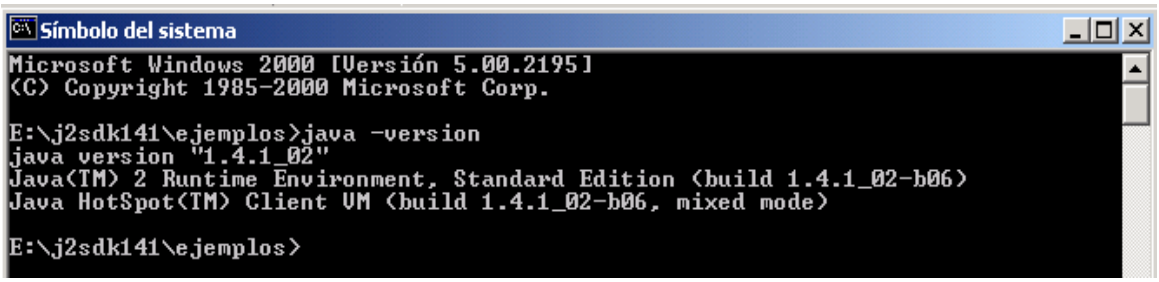


Figura 3.6. Versión de java

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.

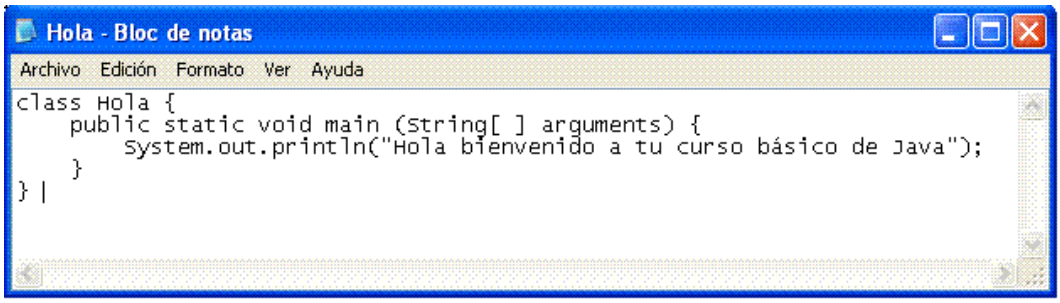


Figura 3.7. Archivo Hola.java

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

3.8.4. Kawa como editor Java

Kawa es un estupendo IDE, potente y de fácil uso. Es un entorno de varias ventanas, en una de ellas muestra en árbol los paquetes Java, su clases, métodos y variables correspondientes. Posibilidad de establecer diversos modos de compilación, soporte para JDBC, resalte de color las palabras reservadas, las clases y muchas cosas más. Una buena y barata alternativa a los entornos más avanzados.
Aquí solo veremos algunos ejemplos muy sencillos para ver la aplicación de Java como un lenguaje de programación.

3.9. Ejercicios y aplicaciones de Java

En esta sección se verán algunos ejemplos de programas realizados con lenguaje Java y compilados con ayuda de Kawa, son programas muy sencillos ya que solo se logro ver desde el inicio del capitulo las características esenciales del lenguaje, así que estos ejemplos son una herramienta útil para comprender el funcionamiento de las diferentes sentencias de Java, el manejo de clases y de métodos de los cuales hablamos anteriormente. También veremos algunos ejemplos de applets que muestran el lado grafico de Java.

"Hola Mundo". Este programa es el más común, como ejemplo en todos los lenguajes de programación, así que mostramos el siguiente código donde se muestra como se hace en Java.

```
/*Hola Mundo*/
import java.io.*;
public class HolaMundo
{
        public static void main (String[] args)
        {
```

```
                System.out.println("Hola Mundo");
        }
}
```

"Pirámide". Este programa, únicamente muestra a partir de un numero, la numeración correspondiente para llegar a el, a partir del numero 1 se despliega una pirámide de números.

```
public class Piramide
{
        public static void main (String[] args)
        {
                int numero;
                int I;
                int j;
                MyInput Entrada = new MyInput();

                numero = Entrada.readInt();

                for (I=1;I<=numero;I++)
                {
                        for (j=1;j<=I;j++)

                 System.out.print(j+" ");
                System.out.print("\n");
                }
        }
}
```

al ejecutar este programa se muestra lo siguiente:

```
10
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
```

"Serie de Fibonacci". Este programa nos muestra la serie de Fibonacci, inicialmente se pide un numero el cual será el fin de la serie, si pertenece a la serie, nos muestra como llegar a el y el ahí termina, sino pertenece a la serie, nos despliega un comentario que nos muestra que ese numero no pertenece a la serie.

```
/*Serie de fibonacci*/
import java.io.*;
import java.util.*;

public class Fibonacci
{
   static private int iresultado;
```

```
   static private int ianterior;
   static private int numero;

        public Fibonacci()
        {
                iresultado=1;
                ianterior=0;
        }
static private StringTokenizer stok;
//Busca un token la aparicion de un caracter dentro de una cadena

static private BufferedReader br =
//Nos crea un buffer de caracteres (flujo de caracteres)

new BufferedReader ( new InputStreamReader(System.in));

        public static int readInt()
        {  //Inicio de leer entero
                int i=0;
                try  //Nos ayuda a saber cuando hay un problema
                {
                        String str = br.readLine(); //Nos lee lo que hay en el buffer
                        StringTokenizer stok = new StringTokenizer(str);
                        i = new Integer (stok.nextToken()).intValue();
                }
                catch (IOException e) //Nos cacho el error
                {
                        System.out.println(e);
                }
                return i;
        } //Fin de Leer Entero


        public static double readDouble()
        { //Inicio de leer
                double i=0;
                try
                {
                        String str = br.readLine();
                        StringTokenizer stok = new StringTokenizer(str);
                        i = new Double (stok.nextToken()).doubleValue();
                }
                catch (IOException e)
                {
                        System.out.println(e);
                }
                return i;
        }
  public void  calcular()
  {
    iresultado = (iresultado + ianterior);
        ianterior = (iresultado - ianterior);
}
   public int YaFinal()
  {
        int ibandera;
        if (iresultado >= numero)
```

```
        {
                ibandera=1;
        }
        else
        {
                ibandera=0;
        }
        return ibandera;
 }
 public void MuestraRes()
 {
   System.out.print(iresultado+" ");
 }
        public static void main (String[] args)
        {
                Fibonacci Fibo = new Fibonacci();
                System.out.print("Escribe el numero final de la serie de Fibonacci a crear:"+ " ");
                numero = Fibo.readInt();

            while (Fibo.YaFinal() != 1)
                {
                        Fibo.MuestraRes();
                        Fibo.calcular();
                }
                if (iresultado > numero)
                {
                        System.out.print("\n");
                         System.out.print("El número:" + numero + " " + "No pertenece a la
Serie de Fibonacci...");
                }
                if (iresultado == numero)
                {
                   Fibo.MuestraRes();
                        System.out.print("\n");
                   System.out.print("Fin de la Serie...");
                }
        }
}
// fin de programa
```

el resultado que se despliega al ejecutar el programa es el siguiente:

Escribe el número final de la serie de Fibonacci a crear:

- *En caso de darle el valor: 1524*

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
El número: 1524 No pertenece a la Serie de Fibonacci...

- *En caso de darle el valor: 13*

1 1 2 3 5 8 13
Fin de la Serie...

“Calculadora”. Este programa maneja un menú, en el cual inicialmente pide dos números, enseguida muestra un menú el cual pide que operación es la que desea realizar, posteriormente muestra el resultado obtenido de la operación seleccionada.

```
/*Calculadora*/
import java.io.*;
import java.util.*;

 public class Operaciones
{
   static private double resultado,a,b;
   static private int operacion;
   static private StringTokenizer stok;
//Busca un token la aparicion de un caracter dentro de una cadena
   static private BufferedReader br =
//Nos crea un buffer de caracteres (flujo de caracteres)
   new BufferedReader ( new InputStreamReader(System.in));

        public static int readInt()
        {   //Inicio de leer entero
                int i=0;
                try  //Nos ayuda a saber cuando hay un problema
                {
                        String str = br.readLine(); //Nos lee lo que hay en el buffer
                        StringTokenizer stok = new StringTokenizer(str);
                        i = new Integer (stok.nextToken()).intValue();
                }
                catch (IOException e) //Nos cacho el error
                {
                        System.out.println(e);
                }
                 return i;
        }       //Fin de Leer Entero
                public static double readDouble()
        {       //Inicio de leer
                double i=0;
                try
                {
                        String str = br.readLine();
                        StringTokenizer stok = new StringTokenizer(str);
                        i = new Double (stok.nextToken()).doubleValue();
                }
                catch (IOException e)
                {
                        System.out.println(e);
                }
                return i;
        }
 public void menu()
 {
  System.out.println("1.- SUMA" + " ");
  System.out.println("2.- RESTA" + " ");
  System.out.println("3.- MULTIPLICACION" + " ");
  System.out.println("4.- DIVISION" + " ");
  System.out.print("Elige una Opcion del MENU:" + " ");
 }
 public void SRMD()
 {
   if (operacion == 1)
```

```
                {
                        System.out.print("\n");
                        resultado = a + b;
                        System.out.print("El resultado es:" + " " + resultado + " " );
                }

                if (operacion == 2)
                {
                   System.out.print("\n");
                        resultado = a - b;
                        System.out.print("El resultado es:" + " " + resultado + " " );
                }
                if (operacion == 3)
                {
                        System.out.print("\n");
                        resultado = a * b;
                        System.out.print("El resultado es:" + " " + resultado + " " );
                }

                if (operacion == 4)
                {
                        System.out.print("\n");
                        resultado = a / b;
                        System.out.print("El resultado es:" + " " + resultado + " " );
                }
 }
        public static void main (String[] args)
        {
                Operaciones num = new Operaciones();
                System.out.print("Escribe el primer número:"+ " ");
                a = num.readDouble();
                System.out.print("Escribe el segundo número:"+ " ");
                b = num.readDouble();
                num.menu();
                operacion=num.readInt();
                num.SRMD();
        }
}
```

El resultado de la compilación de este programa es el siguiente:

```
Escribe el primer número: 5
Escribe el segundo número: 2
1.- SUMA
2.- RESTA
3.- MULTIPLICACION
4.- DIVISION
Elige una Opción del MENU: 3

El resultado es: 10.0
```

3.9.1. Aplicaciones gráficas con applets 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.

Calculadora.java. El applet Calculadora.java presenta una calculadora básica, tomando como herramienta de construcción el AWT. Es una calculadora que realiza las operaciones más habituales, ilustra perfectamente los mecanismos de presentación en pantalla de Componentes del AWT, todos los botones se colocan en posiciones determinadas.

Las operaciones matemáticas no son motivo de estudio, pero sí se podría completar la calculadora con más funciones y hacerla semejante a las que se proporcionan con casi todos los entornos gráficos. La clase **BotonCalculadora** es la fundamental en este ejemplo, ya que nos permite colocar botones en cualquier sitio y con cualquier tamaño.

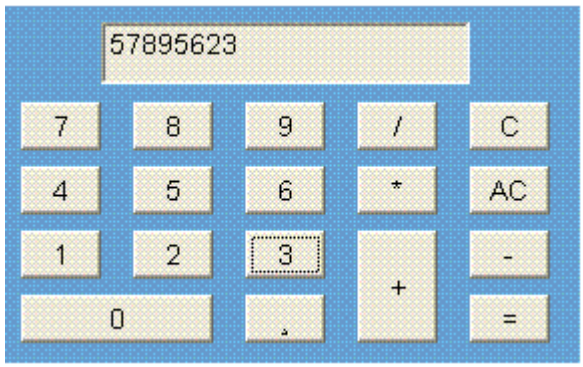


Figura 3.9. Calculadora.java

Aplicaciones de applet para matemáticas y física

Ahora se verán algunos ejemplos de applets que fueron realizados con la finalidad de apoyar a la materia de matemáticas y física, estas gráficas tienen movimiento y se les pueden modificar los valores que se desean. Esto con la fin de hacer el curso más didáctico con ejemplos animados con Java y así lograr que los alumnos se interesen más en la materia.

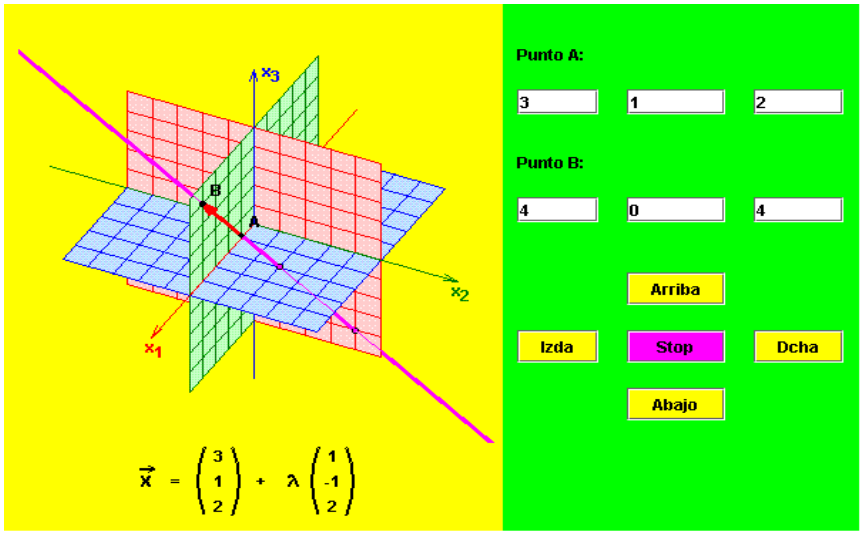


Figura 3.10. Ecuación Vectorial de una Recta en el Espacio.

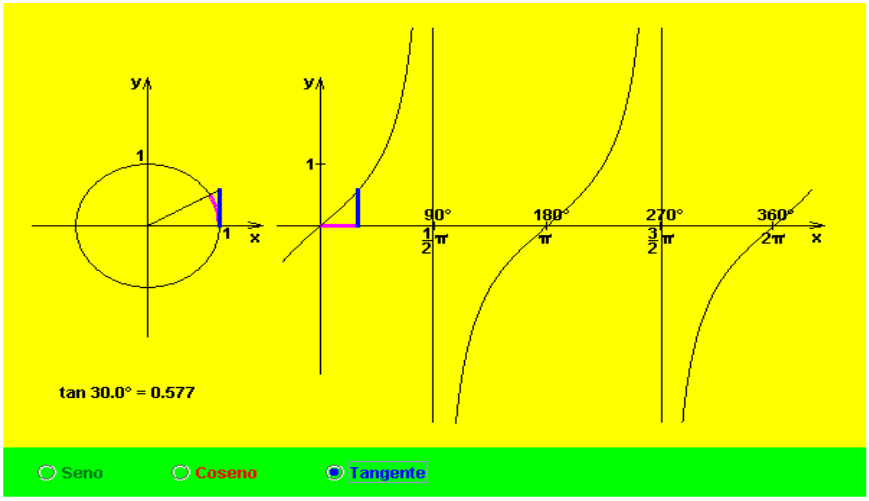


Figura 3.11. Seno, Coseno y tangente de un Angulo.

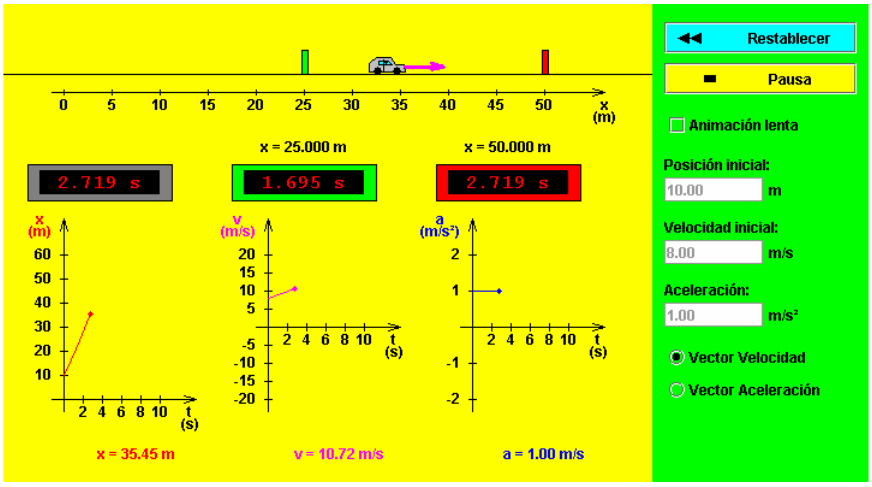


Figura 3.12. Movimiento de Aceleración Constante.

Se ha logrado hacer un resumen de algunas partes del famoso lenguaje Java, 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 logrará algo mejor.

Por lo tanto se verán 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.

Esto último es muy importante ya que la finalidad de este módulo en cuanto a la didáctica, es tratar de involucrar a los alumnos de nivel bachillerato por la programación, así que por lo regular siempre les llama más la atención por medio de imágenes y que mejor que sean animadas, esto se puede hacer con ayuda de Java, y por otro lado también es una buena herramienta para los profesores como apoyo a su materia desarrollando paginas Web con la información necesaria para su materia en curso, como se vio en los ejemplos anteriores java tiene muchas aplicaciones así que ya depende de el programador cúal es la aplicación que necesita desarrollar. Pero 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 4

DIDÁCTICA DE LA PROGRAMACIÓN

4.1. Antecedentes

Se entiende que la capacitación pedagógica básica comprende las bases teóricas, metodológicas y técnicas requeridas para el desarrollo de conocimientos, habilidades, destrezas y competencias específicas que deben ser dominadas para orientar un aprendizaje efectivo.

La educación y la instrucción de las nuevas generaciones es una labor compleja y sutil, la cual se trata, nada menos, que de desarrollar y formar el carácter, la inteligencia y la personalidad de las nuevas generaciones. 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.

Vistas en conjunto, las actividades típicas de un buen profesor se reparten en tres grandes fases: planeamiento, orientación y control.

El ***planeamiento*** se manifiesta es una sucesión de actividades de previsión y de programación de labores escolares que, partiendo de lo más general y sintético, se va particularizando progresivamente y llega a los últimos pormenores concretos sobre los datos informativos que deben enseñarse, los medios auxiliares que han de utilizarse, las actividades y trabajos que los alumnos han de ejecutar, y por fin, sobre las cuestiones que se han de escoger y formular en los exámenes finales.

La técnica docente exige que el profesor prepare conscientemente todas las etapas de su trabajo para conducir bien el aprendizaje de los alumnos.

La ***orientación del aprendizaje*****,** a su vez, se divide en las siguientes fases, destinadas a acompañar cada paso del proceso de aprendizaje:

a) La motivación del aprendizaje. El profesor, por el empleo hábil de técnicas, recursos y procedimientos de estímulo, despierta el interés de los alumnos y desarrolla su gusto por el estudio, consiguiendo de este modo captar su atención e incitarlos al esfuerzo para aprender la materia, efectuando con provecho todos los trabajos programados.

b) La presentación de la materia; mediante ella, el profesor, usando las técnicas, recursos y procedimientos propios de esta fase, hace que los alumnos logren una comprensión inicial del asunto que han de aprender. Esta comprensión inicial tiene una importancia fundamental para el éxito del aprendizaje.

c) La dirección de actividades de los alumnos. Por ella, el profesor lleva a sus alumnos, que ya han comprendido el asunto, a trabajar activamente con los datos o elementos de la materia, de modo que la asimilen; es la fase típica de los trabajos prácticos de aplicación que el profesor debe dirigir con seguridad y técnica adecuadas; es quizás la fase más importante y decisiva para obtener un aprendizaje auténtico.

d) La integración del contenido del aprendizaje, en que el profesor pasa a utilizar procedimientos especiales destinados a integrar en la mente de los alumnos lo que habían aprendido analítica y parcialmente en las fases anteriores. Pretende proporcionarles una visión de conjunto, bien concatenada y ordenada, de todo lo ya aprendido, aclarando sus relaciones y dándoles una perspectiva definida.

e) La fijación del contenido del aprendizaje, con el fin de consolidarlo definitivamente y convertirlo en una conquista permanente del alumno.

La ***verificación*** del aprendizaje se despliega también en cuatro fases típicas y necesarias para la buena dirección del proceso de aprendizaje. Son:

a) El sondeo o exploración y el pronóstico de aprendizaje, verificación preliminar, indispensable de toda labor docente, Consiste en la observación de las condiciones reales en que se encuentran los alumnos al empezar el curso, tanto en lo referente a sus capacidades o carencias específicas, como a su preparación, tanto general como específica, en la asignatura que se va a enseñar. Por más precarias y deficientes que sean estas condiciones, es de ellas de donde ha de partir todo el trabajo constructivo del profesor en la clase. Sobre los hechos consignados por ese sondeo preliminar, el profesor hará el pronóstico de lo que se podrá realizar durante el curso y de los resultados que, entre límites razonables, se pueden esperar de esa clase de alumnos.

b) La conducción de la clase y el control de la disciplina, para asegurar un ambiente de orden y disciplina en las aulas, pues sin él no podrá haber buen rendimiento en el trabajo, y para inculcar en los alumnos los hábitos sociales indispensables en el estudio y en la vida.

c) El diagnóstico y la rectificación del aprendizaje: el profesor auscultará, en forma explícita y periódicamente, las lagunas, dificultades y problemas que los alumnos encuentran en el aprendizaje de la materia. Procurará identificar las causas de esos errores, problemas y dificultades, y orientará a los alumnos para que los rectifiquen y superen, con el fin de que el aprendizaje no resulte imperfecto o incompleto; es la función terapéutica o correctiva del aprendizaje.

d) La comprobación y la evaluación del rendimiento obtenido, para averiguar hasta qué punto el alumno individualmente y la clase en conjunto han conseguido los resultados previstos y deseados. Es la fase típica de pruebas y exámenes. Por ella juzga el profesor no sólo el rendimiento logrado por los alumnos, sino también su propia eficacia como incentivador y orientador de tales alumnos en el aprendizaje de su materia, en una palabra, mide también su propia eficiencia a través de los resultados que ha conseguido en su alumnado.

4.2. Metodologías de la enseñanza

Las metodologías de la enseñanza[11], 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:

a) Ser vividas
b) Ser posibles
c) Estar diversificadas
d) Ser satisfactorias
e) 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. Obedecen a una lógica de desarrollo de los contenidos en tres momentos, introducción, desarrollo y síntesis, si el profesor o los alumnos desean hacer modificaciones o bien sugerir actividades no tipificadas, deberán completar.

11. *Enseñanza*. Es el proceso mediante el cual se comunican o transmiten conocimientos especiales o generales sobre una materia.

- ***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.

4.2.1. 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.

Esto requiere de una rigurosa formación para trabajarlas adecuadamente y con los cuidados necesarios.

4.2.2. Técnicas didácticas

Las técnicas didácticas[12] son el entramado organizado por el docente a través de las cuales pretende cumplir su objetivo.

12. Bells, Miriam. *Técnicas didácticas de capacitación*. p.32.

Son mediaciones a final de cuentas, como mediaciones, tienen detrás una gran carga simbólica relativa a la historia personal del docente: su propia formación social, sus valores familiares, su lenguaje y su formación académica; también forma al docente su propia experiencia de aprendizaje en el aula.

Las técnicas didácticas matizan la práctica docente ya que se encuentran en constante relación con las características personales y habilidades profesionales del docente, sin dejar de lado otros elementos como las características del grupo, las condiciones físicas del aula, el contenido a trabajar y el tiempo.

Las técnicas didácticas forman parte de la didáctica. En este estudio se conciben como el conjunto de actividades que el maestro estructura para que el alumno construya el conocimiento, lo transforme y lo evalúe; además de participar junto con el alumno en la recuperación de su propio proceso. De este modo las técnicas didácticas ocupan un lugar medular en el proceso de enseñanza aprendizaje, son las actividades que el docente planea y realiza para facilitar la construcción del conocimiento.

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.

Las técnicas didácticas que aquí se exponen se agrupan considerando que a cada actividad del profesor le corresponde una actividad del alumno.

Expositivas

- ***Exposición.*** Es un discurso informal 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 platica 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.

4.3. Bases de la evaluación

Evaluar consiste en obtener el juicio de valor de una medición, al comprarla con alguna ley o norma. La evaluación, podría definirse como la interpretación de una medida o medidas en relación a la norma ya establecida.
La evaluación educativa debe ser sistemática, continua e integral. Es sistemática la evaluación cuando obedece a un plan preconcebido (a una programación) y no se hace de modo ocasional o incidental. La evaluación continua , hace referencia al hecho de que la evaluación constituye una etapa más del proceso educativo, por lo que éste debe ser evaluado, asimismo, integral, ya que debe ser evaluados todos los elementos que intervienen en la educación.

4.3.1. Razones que justifican la evaluación

Las principales razones que justifican la evaluación son las siguientes:

a) si el principal valor de la evolución cosiste en permitir una deficiencia de aprendizaje apenas se produce para poder poner remedio inmediato, queda claro que, de no evaluar a los alumnos, éstos avanzarán por el camino de las distintivas enseñanzas sin saber qué fallos van teniendo o qué lagunas se les van formando, con lo que llegará un momento en que los nuevos aprendizajes no podrían realizarse por faltarles la base necesaria, o carecerían de la necesaria consistencia.

b) por eso la evaluación no debe tener un carácter selectivo, tal como se viene considerando en determinados sectores educativos, sino que debe tener un sentido de ayuda u orientación constante del alumno.

c) la evaluación de los resultados del aprendizaje nos proporcionan conocimientos de lo que el alumno ha rendido en la relación a los niveles objetivos, por una parte, y a sus actitudes personales, por otra. Efectivamente, la evaluación nos indica si cada escolar va superando los objetivos legalmente establecidos y que la sociedad exige a sus miembros.

d) mediante los resultados de la evaluación podemos analizar las causas que pudieron haber motivado deficiencias en el logro de los objetivos propuestos, y en consecuencia podrá poner resoluciones para paliar dichas deficiencias.

e) la evaluación nos ayuda a aprender de la experiencia y ya no incurri en el futuro en los mismos errores. En definitiva, la evaluación contribuye a la constante reelaboración de la estrategia docente, e impide la fijación de pautas rígidas e inmovibles en la conducción del proceso educativo.

f) la comprobación del rendimiento que origina la evaluación favorece el agrupamiento de alumnos, sirve para promoverlos de nivel o pases, ayuda a descubrir aptitudes, intereses y preguntas vacacionales, identificar diferencias individuales y, en fin, favorece la personalización en el tratamiento educativo.

g) la evaluación permite poseer un conocimiento inicial de las capacidades del alumno lo más exacto posible, al estar apoyado en instrumentos diagnóstico tipificados.

h) con la evaluación se asignan las calificaciones de forma objetiva, al utilizar instrumentos de medida científicos.

4.3.2. 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.

4.3.3. 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 diagnóstico

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

a) Diagnóstico 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.

b) 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 diagnóstico 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 más motivador que el 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.

4.4. Elementos de didáctica y manejo de grupos

Podemos analizar seis elementos fundamentales en el proceso enseñanza-aprendizaje: el alumno, el profesor, los objetivos, la materia, las técnicas de enseñanza y el entorno social, cultural y económico en el que se desarrolla.

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. Los objetivos sirven de guía en el proceso, y son formulados al inicio de la programación docente. La materia, por su parte, constituye la sustancia, el conocimiento que es necesario transmitir de profesor a alumno, y que debe ser asimilada por éste. Constituyen las técnicas de enseñanza, los medios y métodos a través de los cuales realizaremos la labor docente. Por último, el entorno condiciona en gran medida el proceso.

Por tanto, la enseñanza y el aprendizaje son dos fenómenos correlativos y relacionados por lo que se denomina la relación didáctica. Se distinguen tres etapas en la acción didáctica:

A) Planteamiento. En esta etapa se formulan los objetivos educativos y los planes de trabajo adaptados a los objetivos previstos. La formulación de un plan implica la toma de decisiones anticipada y la reflexión con anterioridad a la puesta en práctica.

B) Ejecución. Posteriormente al planteamiento, el profesor pone en práctica los recursos y métodos didácticos, desarrollándose el proceso de enseñanza.

C) Evaluación. Es la etapa en la que se verifican los resultados obtenidos con la ejecución, materializándose en el proceso de evaluación.

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 líneas 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

Mientras que en la lección magistral el alumno tiene un comportamiento básicamente pasivo, en las técnicas de trabajo en grupo debe participar de modo activo. Al trabajar en grupo, el alumno puede resolver problemas prácticos, aplicar conocimientos teóricos y también recibir orientación por parte del profesor.

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:

a) Lograr la individualización de la enseñanza.
b) Conseguir la participación activa de todos los alumnos en el proceso de enseñanza-aprendizaje.
c) Desarrollar la habilidad de trabajar en equipo.
d) 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.

La correcta aplicación del método suele requerir un número limitado de alumno en cada grupo de trabajo pues los grupos excesivamente grandes dificultan la colaboración y la participación activa de todos los alumnos. La labor del profesor es orientadora y motivadora del proceso de trabajo de los estudiantes.

Además de los objetivos generales, el método de trabajo en grupos permite la consecución de objetivos específicos. En este sentido si nos centramos en una disciplina concreta, con el trabajo en grupo se pueden alcanzar las siguientes finalidades:

a) Desarrollar la capacidad crítica autónoma al enfrentar el alumno con una situación problemática.
b) Conseguir desarrollar las habilidades de expresión oral y escrita.
c) Aplicar lo aprendido.
d) Obtener por parte del profesor información continúa sobre el desarrollo del aprendizaje, la comunicación se facilita por la conversación entre los miembros del grupo y su diversidad de opiniones.

Tanto los miembros integrantes de los grupos como el tema objeto del trabajo puede ser impuesto por el profesor o elegidos por los propios alumnos. El permitir cierto margen de libertad en la elección del tema objeto del trabajo mejora la motivación y el interés de los estudiantes.

4.4.1. Motivación en cómputo

Principio de motivación. Este aspecto es crucial, nadie aprende si no le mueve alguna razón.

a) *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.

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

c) *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.

d) *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.

e) *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.

4.4.2. Recomendaciones para la motivación

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[13]:

- 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[14]

- 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 más 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.

13. Peñaloza Romero Ernesto. *Didáctica de la programación*. p. 37.

14.*Autoaprendizaje*. Es la forma de aprender principalmente por uno mismo. Consiste en aprender buscando uno mismo la información, haciendo prácticas o experimentos. A una persona que aprende por sí misma se le llama autodidacta.

- Arredondo, M. *Notas para un modelo de docencia: Formación pedagógica de profesores universitarios. Teoría y experiencias en México*. México (1998) ANUIES-UNAM. CESU.

CAPÍTULO 5

LENGUAJE PHP Y APLICACIONES WEB

5.1. Antecedentes

El lenguaje HTML no es lenguaje de programación sino, se trata de un lenguaje descriptivo, es un lenguaje de interpretación que tiene como objeto dar formato al texto y las imágenes que pretendemos visualizar en el navegador.

Con ayuda de este lenguaje se puede introducir enlaces, seleccionar el tamaño de la fuente o intercalar imágenes, todo esto de una manera prefija y en ningún caso inteligente. En efecto, el HTML no permite realizar un simple cálculo matemático o crear una página de la nada a partir de una base de datos. A decir verdad, HTML aunque muy útil a pequeña escala, resulta bastante limitado a la hora de concebir grandes sitios o portales.

Es esta deficiencia HTML la que ha hecho necesario el empleo de otros lenguajes mucho más versátiles y de un aprendizaje relativamente más complicado, capaces de responder de manera inteligente a las demandas del navegador y que permiten la automatización de determinadas tareas tediosas e irremediables como pueden ser las actualizaciones, el tratamiento de pedidos de una tienda virtual por dar un ejemplo.

Estos lenguajes capaces de recrear a partir de ciertos "scripts" un sin fin de páginas automatizadas son los protagonistas de este concepto de páginas dinámicas.

Así que para lograr desarrollar páginas dinámicas con HTML usaremos las aplicaciones del Lenguaje de programación PHP.

5.2. Lenguaje de programación PHP

PHP ha recorrido un largo camino en los últimos años. Crecer hasta ser uno de los más importantes lenguajes de programación en entornos Web no ha sido tarea fácil.

PHP, acrónimo de “Hypertext Preprocessor", es un lenguaje "Open Source" es decir, es un lenguaje de código abierto, interpretado de alto nivel, especialmente pensado para desarrollos web y el cual puede ser incluido en páginas HTML. La mayoría de su sintaxis es similar a C, Java y Perl y es fácil de aprender. La meta de este lenguaje es permitir crear páginas dinámicas de una manera rápida y fácil, aunque se pueda hacer mucho más con PHP.

Una respuesta corta y concisa, pero, Un ejemplo nos aclarará las cosas:

```
<html>
  <head>
      <title>Ejemplo</title>
  </head>
    <body>
      <?php
      echo "Hola, &iexcl;soy un script PHP!";
      ?>
   </body>
</html>
```

Puede apreciarse que no es lo mismo que un script escrito en otro lenguaje de programación como en C. En vez de escribir un programa con muchos comandos para crear una salida en HTML, escribimos el código HTML con cierto código PHP embebido (incluido) en el mismo, que producirá cierta salida (en nuestro ejemplo, producirá un texto). El código PHP se incluye entre etiquetas especiales de comienzo y final que nos permitirán entrar y salir del modo PHP.

Lo que distingue a PHP de la tecnología Javascript, la cual se ejecuta en la máquina cliente, es que el código PHP es ejecutado en el servidor. Si se tuviera un script similar al de nuestro ejemplo en nuestro servidor, el cliente solamente recibiría el resultado de su ejecución en el servidor, sin ninguna posibilidad de determinar qué código ha producido el resultado recibido. El servidor web puede ser incluso configurado para que procese todos los archivos HTML con PHP. En la Figura 5.1 se muestra como es que trabajan las páginas que tiene integrado código PHP con el entorno de Internet.

Figura 5.1. Modo de trabajo de PHP

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 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. Y esto no es todo, se puede hacer mucho más.

PHP puede ser utilizado en cualquiera de los principales sistemas operativos del mercado, incluyendo Linux, muchas variantes Unix (incluyendo HP-UX, Solaris y OpenBSD), Microsoft Windows, Mac OS X, RISC OS y probablemente alguno más. 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 aquellos otros que soporten el estándar CGI, PHP puede usarse como procesador CGI.

De modo que, con PHP se tiene la libertad de elegir el sistema operativo y el servidor de su gusto. También 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 grandes (incluyendo la biblioteca PEAR) están escritas íntegramente usando programación orientada a objetos.

Con PHP no se encuentra limitado a resultados en HTML. Entre las habilidades de PHP se incluyen: creación de imágenes, archivos PDF y películas Flash. También puede presentar otros resultados, como XHTM y archivos XML. PHP puede autogenerar estos archivos y almacenarlos en el sistema de archivos en vez de presentarlos en la pantalla.

Quizás la característica más potente y destacable de 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	Ingres	Oracle (OCI7 and OCI8)
dBase	InterBase	Ovrimos
Empress	FrontBase	PostgreSQL
FilePro (read-only)	mSQL	Solid
Hyperwave	Direct MS-SQL	Sybase
Informix	ODBC	Unix dbm

Las páginas web que utilizan PHP son tratadas como páginas de HTML comunes y corrientes, y pueden crearlas y editarlas de la misma manera que lo hace con documentos normales de HTML.

5.2.1. 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 raya. En nuestro caso, una letra es a-z, A-Z, y los caracteres ASCII del 127 al 255, como se muestra en el ejemplo siguiente:

```
<?php
$var = "Bob";
$Var = "Joe";
echo "$var, $Var"; // salida "Bob, Joe"
```

PHP ofrece otra forma de asignar valores a las variables: *asignar por referencia*. Esto significa que la nueva variable simplemente referencia (en otras palabras, "se convierte en un alias de" ó "apunta a") la variable original. Los cambios a la nueva variable afectan a la original, y viceversa. Esto también significa que no se produce una copia de valores; por tanto, la asignación ocurre más rápidamente. De cualquier forma, cualquier incremento de velocidad se notará sólo en los bucles críticos cuando se asignen grandes matrices u objetos.

Para asignar por referencia, simplemente se antepone un ampersandsigno "&" al comienzo de la variable cuyo valor se está asignando (la variable fuente). Por ejemplo, el siguiente trozo de código produce la salida 'Mi nombre es Bob' dos veces:

```
<?php
$foo = 'Bob';  // Asigna el valor 'Bob' a $foo
$bar = &amp;$foo;  // Referencia $foo v&iacute;a $bar.
$bar = "Mi nombre es $bar";  // Modifica $bar...
echo $foo;  // $foo tambi&eacute;n se modifica.
echo $bar;
?>
```

Existen mucho más tipos para asignación de variables, pero por el momento sólo se verán los principales y los que utilizaremos al largo del proyecto.

5.2.2. Operadores

Un operador es algo a lo que 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.

Operadores aritméticos

Son los operadores simples que podemos manejar dentro del lenguaje PHP, en la Tabla 5.1 se muestran estos operadores.

Ejemplo	Nombre	Resultado
-$a	Negación	El opuesto de $a.
$a + $b	Adición	Suma de $a y $b.
$a - $b	Substracción	Diferencia entre $a y $b.
$a * $b	Multiplicación	Producto de $a y $b.
$a / $b	División	Cociente de $a y $b.
$a % $b	Módulo	Resto de $a dividido por $b.

Tabla 5.1. Operadores aritméticos.

El operador de división ("/") devuelve un valor flotante en todos los casos, incluso si los dos operandos son enteros (o cadenas que son convertidas a enteros).

Operadores de asignación

El operador básico de asignación es "=". A primera vista, usted podría pensar en él como "es igual a". No lo haga. Lo que quiere decir en realidad es que el operando de la izquierda recibe el valor de la expresión a la derecha (es decir, "se define a").

El valor de una expresión de asignación es el valor que se asigna. Es decir, el valor de "$a = 3" es 3. Esto le permite hacer una que otra cosa curiosa:

Operadores de comparación

Los operadores de comparación, como su nombre indica, le permiten comparar dos valores. En la Tabla 5.2. se muestran los diferentes operadores que se pueden utilizar en PHP.

Ejemplo	Nombre	Resultado
$a == $b	Igual	**TRUE** si $a es igual a $b.
$a === $b	Idéntico	**TRUE** si $a es igual a $b, y son del mismo tipo. (A partir de PHP 4)
$a != $b	Diferente	**TRUE** si $a no es igual a $b.
$a <> $b	Diferente	**TRUE** si $a no es igual a $b.
$a !== $b	No idénticos	**TRUE** si $a no es igual a $b, o si no son del mismo tipo. (A partir de

Ejemplo	Nombre	Resultado
		PHP 4)
$a < $b	Menor que	**TRUE** si $a es escrictamente menor que $b.
$a > $b	Mayor que	**TRUE** si $a es estrictamente mayor que $b.
$a <= $b	Menor o igual que	**TRUE** si $a es menor o igual que $b.
$a >= $b	Mayor o igual que	**TRUE** si $a es mayor o igual que $b.

Tabla 5.2. Operadores de Comparación.

5.3. Estructuras de control

Todo script PHP se compone de una serie de sentencias. Una sentencia puede ser una asignación, una llamada a función, un bucle, una sentencia condicional e incluso una sentencia que no haga nada (una sentencia vacía). Las sentencias normalmente acaban con punto y coma. Además, las sentencias se pueden agrupar en grupos de sentencias encapsulando un grupo de sentencias con llaves. Un grupo de sentencias es también una sentencia. En este capítulo se describen los diferentes tipos de sentencias.

5.3.1. Sentencia IF

La construcción *if* es una de las más importantes características de muchos lenguajes, incluido PHP. Permite la ejecución condicional de fragmentos de código. PHP caracteriza una estructura *if* que es similar a la de C:

```
<?php
if (expr)
   sentencia
?>
```

El siguiente ejemplo mostraría *a* es mayor que *b* si *$a* fuera mayor que *$b*:

```
<?php
if ($a > $b)
  print "a es mayor que b";
?>
```

A menudo, se desea tener más de una sentencia ejecutada de forma condicional. Por supuesto, no hay necesidad de encerrar cada sentencia con una cláusula *if*. En vez de eso, se pueden agrupar varias sentencias en un grupo de sentencias. Por ejemplo, este código mostraría a es mayor que b si *$a* fuera mayor que *$b*, y entonces asignaría el valor de *$a* a *$b*:

```
<?php
 if ($a > $b) {
```

```
    print "a es mayor que b";
    $b = $a;
}
?>
```

Las sentencias *if* se pueden anidar indefinidamente dentro de otras sentencias *if*, lo cual proporciona una flexibilidad completa para ejecuciones condicionales en las diferentes partes de tu programa.

A menudo se requiere ejecutar una sentencia si se cumple una cierta condición, y una sentencia distinta si la condición no se cumple. Esto es para lo que sirve *else*. *else* extiende una sentencia *if* para ejecutar una sentencia en caso de que la expresión en la sentencia *if* se evalúe como **FALSE**. Por ejemplo, el siguiente código mostraría a es mayor que b si *$a* fuera mayor que *$b*, y a NO es mayor que b en cualquier otro caso:

```
<?php
 if ($a > $b) {
    print "a es mayor que b";
 } else {
    print "a NO es mayor que b";
 }
?>
```

elseif, como su nombre sugiere, es una combinación de *if* y *else*. Como *else*, extiende una sentencia *if* para ejecutar una sentencia diferente en caso de que la expresión *if* original se evalúa como **FALSE**. No obstante, a diferencia de *else*, ejecutará esa expresión alternativa solamente si la expresión condicional *elseif* se evalúa como **TRUE**. Por ejemplo, el siguiente código mostraría a es mayor que b, a es igual a b o a es menor que b:

```
<?php
 if ($a > $b) {
    print "a es mayor que b";
 } elseif ($a == $b) {
    print "a es igual que b";
 } else {
    print "a es mayor que b";
```

5.3.2. Sentencia WHILE

Los bucles *while* son los tipos de bucle más simples en PHP. Se comportan como su contrapartida en C. La forma básica de una sentencia *while* es:

while (expr) sentencia

El significado de una sentencia *while* es simple. Le dice a PHP que ejecute la(s) sentencia(s) anidada(s) repetidamente, mientras la expresión *while* se evalúe como **true**. El valor de la expresión es comprobado cada vez al principio del bucle, así que incluso si este valor cambia durante la ejecución de la(s)

sentencia(s) anidada(s), la ejecución no parará hasta el fin de la iteración (cada vez que PHP ejecuta las sentencias en el bucle es una iteración). A veces, si la expresión *while* se evalúa como **false** desde el principio de todo, las sentencias anidadas no se ejecutarán ni siquiera una vez.

Como con la sentencia *if*, se pueden agrupar multiples sentencias dentro del mismo bucle *while* encerrando un grupo de sentencias con llaves, o usando la sintaxis alternativa:

while (expr): sentencia ... endwhile;

Los bucles *do..while* son muy similares a los bucles *while*, excepto que las condiciones se comprueban al final de cada iteración en vez de al principio. La principal diferencia frente a los bucles regulares *while* es que se garantiza la ejecución de la primera iteración de un bucle *do..while* (la condición se comprueba sólo al final de la iteración), mientras que puede no ser necesariamente ejecutada con un bucle *while* regular (la condición se comprueba al principio de cada iteración, si esta se evalúa como **false** desde el principio la ejecución del bucle finalizará inmediatamente).

Hay una sola sintaxis para los bucles *do..while*:

```
<?php
 $i = 0;
 do {
    print $i;
} while ($i>0);
?>
```

El bucle de arriba se ejecutaría exactamente una sola vez, después de la primera iteración, cuando la condición se comprueba, se evalúa como **false** ($i no es más grande que 0) y la ejecución del bucle finaliza.

5.3.3. Sentencia FOR

Los bucles *for* son los bucles más complejos en PHP. Se comportan como su contrapartida en C. La sintaxis de un bucle *for* es:

for (expr1; expr2; expr3) sentencia

La primera expresión (*expr1*) se evalúa (ejecuta) incondicionalmente una vez al principio del bucle.

Al comienzo de cada iteración, se evalúa *expr2* . Si se evalúa como **true**, el bucle continúa y las sentencias anidadas se ejecutan. Si se evalúa como **false**, la ejecución del bucle finaliza.

5.3.4. Sentencia SWITCH

La sentencia *switch* es similar a una serie de sentencias *if* en la misma expresión. En muchas ocasiones, se quiere comparar la misma variable o expresión con muchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual. Para ello sirve la sentencia *switch*.

En este caso se debe de tener en cuenta que al contrario que otros lenguajes de programación, *continue* se aplica a switch y funciona de manera similar a *break*. Si se tiene un switch dentro de un bucle y se desea continuar con el paso siguiente en el bucle externo, se usará *continue 2*.

Los siguientes dos ejemplos son dos modos distintos de escribir la misma cosa, uno usa una serie de sentencias *if*, y el otro usa la sentencia *switch*:

```
<?php
if ($i == 0) {
   print "i equals 0";
} elseif ($i == 1) {
   print "i equals 1";
} elseif ($i == 2) {
   print "i equals 2";
}

switch ($i) {
   case 0:
      print "i equals 0";
      break;
   case 1:
      print "i equals 1";
      break;
   case 2:
      print "i equals 2";
      break;
}
?>
```

5.4. Funciones

Muchas veces, cuando se trabajo en el desarrollo de una aplicación, surge la necesidad de ejecutar un mismo bloque de código en diferentes partes de nuestra aplicación. Una **Función** no es más que un bloque de código al que se le pasa una serie de parámetros y devuelve un valor. Como todos los lenguajes de programación, PHP trae una gran cantidad de funciones, pero las funciones más importantes son las que nosotros creamos.
Para declara una función se debe utilizar la instrucción **function** seguido del nombre que le vamos a dar, y después entre parentesis la lista de argumentos separados por comas, aunque también habrá funciones que no recojan ningún argumento.

```
function nombre_de_funcion (arg_1, arg_2, ..., arg_n)
{
    bloque de código
}
```

Cualquier instrucción válida de PHP puede aparecer en el cuerpo (lo que antes hemos llamado bloque de código) de una función, incluso otras funciones y definiciones de clases.

Parámetros de las funciones

Existen dos formas de pasar los parámetros a una función, por **valor** o por **referencia**.
Cuando se pasa una variable por **valor** a una función, ocurra lo que ocurra en ésta en nada modificará el contenido de la variable. Mientras que si lo hacemos por **referencia**, cualquier cambio acontecido en la función sobre la variable lo hará para siempre.
E variable lo hará para siempre.
En PHP, por defecto, las variables se pasan por valor. Para hacerlo por referencia debemos anteponer un **ampersand** (**&**) a la variable.

```
<?php
function suma ($x, $y)
{
    $x = $x + 1;
    return $x+$y;
}
$a = 1;
$b = 2;

//parámetros por valor
echo suma ($a, $b);    // imprimirá 4
echo $a;    // imprimirá 1

//parámetros por referencia
echo suma (&$a, $b);    // imprimirá 4
echo $a;     //imprimirá 2
?>
```

Si se requiere que un parámetro de una función se pase siempre por referencia se debe anteponer un ampersand (**&**) al nombre del parámetro en la definición de la función.

Como ahorrar líneas de código

En las lecciones anteriores se ha aprendido el uso básico de las funciones de PHP para trabajar con MySQL. En esta lección y sucesivas se van a ver nuevas funciones que facilitan y dan potencia a las páginas Web.
Por lo general, todos los script tienen partes de código iguales, las funciones **include()** y **require()** ahorrar muchas de estas líneas de código. Ambas funciones hacen una llamada a un determinado fichero pero de dos maneras diferentes, con **include()**, se inserta lo que contenga el fichero que llamemos

de manera literal en nuestro script, mientras que con **require()**, se dice que el script necesitará parte de código de se encuentra en el fichero que llama **require()**.
Se muestran algunos códigos para aclarar mejor las cosas:

```
<?php
include ("header.inc");
echo "Hola Mundo";
include ("footer.inc");
?>
```

Si se tiene en cuenta que el fichero **header.inc** contiene:

```
<html>
<body>
```

y el fichero **footer.inc** contiene:

```
</body>
</html>
```

El script sería equivalente a:

```
<html>
<body>
<?php
<?php
echo "Hola Mundo";
?>
</body>
</html>
```

Ahora el script de ejemplo para la función **require()**:

```
<?php
require ("config.inc");
include ("header.inc");
echo $cadena;
include ("footer.inc");
?>
```

Donde el fichero **config.inc** tendría algo como esto:

```
<?php
$cadena = "Hola Mundo";
```

5.5. Clases

Las Clases son máximo exponente de la Programación Orientada a Objetos (POO). PHP no es un lenguaje orientado a objetos, pero implementa las características que permiten definir las clases.

Las clases sirven para hacer el código más legible, y lo que es más importante, reutilizable. Escribir una clase es sin duda más largo que escribir el código directamente, pero a la larga es más rentable por su portabilidad a otras, pero a la larga es más rentable por su portabilidad a otras aplicaciones y su mantenimiento.

Las Clases no son más que una serie de variables y funciones que describen y actúan sobre algo. Por ejemplo, se va a crear la clase **automóvil**, la cual tendrá diversas variables, **$color**, **$modelo**, **$marca**, **$potencia**, **$matricula** y habrá una serie de funciones que actuarán sobre la clase **automóvil** como **Precio()**, **Acelerar()**, **Frenar()**, **Girar()**y **Reparar()**.

Como ejemplo se crea la clase **mysql**, que servirá para realizar consultas a las bases de datos MySQL.

```
<?php
class DB_mysql {

/* variables de conexión */
var $BaseDatos;
var $Servidor;
var $Usuario;
var $Clave;

/* identificador de conexión y consulta */
var $Conexion_ID = 0;
var $Consulta_ID = 0;

/* número de error y texto error */
var $Errno = 0;
var $Error = "";

/* Método Constructor: Cada vez que creemos una variable
de esta clase, se ejecutará esta función */
function DB_mysql($bd = "", $host = "localhost", $user = "nobody", $pass = "") {
      $this->BaseDatos = $bd;
      $this->Servidor = $host;
      $this->Usuario = $user;
      $this->Clave = $pass;
}

/*Conexión a la base de datos*/
function conectar($bd, $host, $user, $pass){

      if ($bd != "") $this->BaseDatos = $bd;
      if ($host != "") $this->Servidor = $host;
      if ($user != "") $this->Usuario = $user;
      if ($pass != "") $this->Clave = $pass;

      // Conectamos al servidor
      $this->Conexion_ID = mysql_connect($this->Servidor, $this->Usuario, $this->Clave);
      if (!$this->Conexion_ID) {
            $this->Error = "Ha fallado la conexión.";
            return 0;
      }
```

```
        //seleccionamos la base de datos
        if (!@mysql_select_db($this->BaseDatos, $this->Conexion_ID)) {
                $this->Error = "Imposible abrir ".$this->BaseDatos ;
                return 0;
        }

        /* Si hemos tenido éxito conectando devuelve
        el identificador de la conexión, sino devuelve 0 */
        return $this->Conexion_ID;
}

/* Ejecuta un consulta */
function consulta($sql = ""){

        if ($sql == "") {
                $this->Error = "No ha especificado una consulta SQL";
                return 0;
        }

        //ejecutamos la consulta
        $this->Consulta_ID = @mysql_query($sql, $this->Conexion_ID);

        if (!$this->Consulta_ID) {
                $this->Errno = mysql_errno();
                $this->Error = mysql_error();
        }
        /* Si hemos tenido éxito en la consulta devuelve
        el identificador de la conexión, sino devuelve 0 */
        return $this->Consulta_ID;
}

/* Devuelve el número de campos de una consulta */
function numcampos() {
                return mysql_num_fields($this->Consulta_ID);
}

/* Devuelve el número de registros de una consulta */
function numregistros(){
                return mysql_num_rows($this->Consulta_ID);
}

/* Devuelve el nombre de un campo de una consulta */
function nombrecampo($numcampo) {
                return mysql_field_name($this->Consulta_ID, $numcampo);
}

/* Muestra los datos de una consulta */
function verconsulta() {

        echo "<table border=1>\n";

        // mostramos los nombres de los campos
        for ($i = 0; $i < $this->numcampos(); $i++){
                echo "<td><b>".$this->nombrecampo($i)."</b></td>\n";
        }
        echo "</tr>\n";
        // mostrarmos los registros
```

```
        while ($row = mysql_fetch_row($this->Consulta_ID)) {
            echo "<tr> \n";
            for ($i = 0; $i < $this->numcampos(); $i++){
                echo "<td>".$row[$i]."</td>\n";
            }
            echo "</tr>\n";
        }
    }

    } //fin de la Clse DB_mysql
    ?>
```

Como se observo, para crear una clase se utiliza la sentencia **class**, y además se creó una función con el mismo nombre que la clase, a esa función se le llama ***constructor*** y se ejecutará cada vez que definamos una variable de esa clase. No es obligatoria la variable de esa clase. No es obligatorio crear un **constructor** en una definición de clase.

Otra cosa importante en las clases es el operador **->**, con el que indicamos una variable o método (parte derecha del operador) de una clase (parte izquierda del operador). Para hacer referencia a la clase que estamos creando dentro de su definición, debemos utilizar **this**.

Y ahora un ejemplo de la clase que se ha creado, y suponiendo que el código anterior se guardo en un fichero llamado **clase_mysql.inc.php**.

```
<body>
<html>
<?php
    require ("clase_mysql.inc.php");
    $miconexion = new DB_mysql ;
    $miconexion->conectar("mydb", "localhost", "nobody", "");
    $miconexion->consulta("SELECT * FROM agenda");
    $miconexion->verconsulta();
?>
</body>
</html>
```

5.6. Procesado de formularios

El lenguaje PHP proporciona una manera sencilla de manejar formularios, permitiéndonos de esta manera procesar la información que el usuario ha introducido. Al diseñar un formulario se debe indicar la página PHP que procesará el formulario, así como en método por el que se le pasará la información a la página

Los Formularios no forman parte de PHP, sino del lenguaje estándar de Internet, HTML, así que lo que viene a continuación es HTML y no PHP.

Todo formulario comienza con la etiqueta ***<FORM ACTION="lo_que_sea.php" METHOD="post/get">***. Con ***ACTION*** indica a el script que va procesar la información que recoge el formulario, mientras que ***METHOD*** indica si el usuario del formulario va ha enviar datos (**post**) o recogerlos (**get**). La etiqueta ***<FORM>*** indica el final del formulario.

A partir de la etiqueta ***<FORM>*** vienen los campos de entrada de datos que pueden ser:

Cuadro de texto:

```
<input type="text" name="nombre" size="20" value="jose">
```

Cuadro de texto con barras de desplazamiento:

```
<textarea rows="5" name="descripcion" cols="20">Es de color rojo</textarea>
```

Casilla de verificación:

```
<input type="checkbox" name="cambiar" value="ON">
```

Botón de opción:

```
<input type="radio" value="azul" checked name="color">
```

Menú desplegable:

```
<select size="1&qu class="codigo"><select size="1" name="dia">
<option selected value="lunes">lunes</option>
<option>martes</option>
<option value="miercoles">miercoles</option>
</select>
```

Boton de comando:

```
<input type="submit" value="enviar" name="enviar">
```

Campo oculto:

```
<input type="hidden" name="edad" value="55">
```

Este último tipo de campo resulta especialmente útil cuando se requiere pasar datos ocultos en un formulario.

Como se habrá observado todos lo tipos de campo tienen un modicador llamado **name**, que no es otro que el nombre de la variable con la cual recogeremos los datos en el script indicado por el modificador **action** de la etiqueta **form,** con **value** establecemos un valor por defecto.

A continuación un ejemplo, para lo cual se crea un formulario en HTML como el que sigue y se le nombra **formulario.html**, el cual va a servir para ejemplificar el funcionamiento de PHP.

formulario.html

```
<HTML>
 <HEAD> </HEAD>
  <BODY BGCOLOR="336699"><FONT face='verdana' color='yellow'><center> <H2>
INSCRIPCIONES A CURSOS DE COMPUTACION </center></H2>
  <P><p><p><HR size=5>
   <FONT face="verdana" color='yellow'>
   <FORM action=Inscripciones.php method='post'<p><p>
     Nombre    :<input type='text' name='nombre' size='50' maxlenght='50'>
     <p>
     Direccion:<input type='text' name='direccion' size='50' maxlenght='30'>
     <p>
     Telefono :<input type='text' name='telefono' size='12' maxlenght='12'>
     <p>
     @-mail     :<input type='text' name='correo' size='20' maxlenght='20'>
     <p>

     Genero: F<input type='radio' checked name='genero' value='F'>
           M<input type='radio' name='genero' value='M'>
     <p>

     Plantel:<select name='plantel' size='1'>
           <option value='1'> Prepa 1 </option>
           <option value='2'> Prepa 2 </option>
           <option value='3'> Prepa 3 </option>
           <option value='4'> Prepa 4 </option>
           <option value='5'> Prepa 5 </option>
           <option value='6'> Prepa 6 </option>
           <option value='7'> Prepa 7 </option>
           <option value='8'> Prepa 8 </option>
           <option value='9'> Prepa 9 </option>
           <option value='10'> CCH Sur </option>
           <option value='11'> CCH Oriente </option>
           <option value='12'> CCH Vallejo </option>
           <option value='13'> CCH Nahucalpan </option>
           <option value='14'> CCH Cuahutitlan </option>
           <p>
           </select> <p>

    <FONT COLOR='WHITE'><H3>CURSOS <HR width=20% align=left></FONT></H3>
    <input type='checkbox' name='asig 1' value='1'>Windows XP
    <BR>
    <input type='checkbox' name='asig 2' value='2'>Word XP
    <BR>
    <input type='checkbox' name='asig 3' value='3'>Excel XP
    <BR>
    <input type='checkbox' name='asig 4' value='4'>Power Point XP
    <BR>
    <input type='checkbox' name='asig 5' value='5'>Photoshop XP
    <BR>
    <input type='checkbox' name='asig 6' value='6'>Corel Draw
    <BR>
    <input type='checkbox' name='asig 7' value='7'>AutoCat 3D
    <BR>
    <input type='checkbox' name='asig 8' value='8'>HTML
    <BR>
    <input type='checkbox' name='asig 9' value='9'>PHP
```

```
    <BR>
    <input type='checkbox' name='asig 10' value='10'>ASP
    <BR>
    <P>
    <input type='submit' value='enviar'>
    <input type='reset' value='limpiar'>
</BODY>
</FORM>
</HTML>
```

El script PHP llamado desde le formulario **Inscripciones.php,** para crear este archivo se editara con ayuda de el block de notas, al igual que se hizo con el archivo formulario.html, solo que en el caso de *Inscripciones* lo se guarda con extensión ***.php*** para que genere el archivo:

Inscripciones.php

```
<HTML>
<HEAD> </HEAD>
  <BODY BGCOLOR =”336699”><FONT face=”verdana’ color=’yellow’><center><H2>
ALUMNOS INSCRITOS </center></H2>
<p> <HR  size=5>
<?php
print “<FONT face=’verdana’ color=#FF33CC><H3> Su nombre es: $nombre”;
print “<Su dirección es: $ direccion;
print “<Su teléfono es: $telefono;
print “< Su E-mail es: $correo;

if ($genero==F)
 {
   print”<FONT face=’verdana’ color=#FF33CC><h3> Su genero es:</FONT></H3> Mujer”;
 }
elseif ($genero==M)
 {
   print “<FONT face=’verdana’ color=#FF33CC><H3> Su genero es: </FONT></H3>Hombre2;
 }
//Variable plantel

if ($plantel==1)
 {
   print “<FONT face=’verdana’ color=#FF33CC><H3> El plantel es: </FONT></H3> Prepa 1”;
 }
if ($plantel==2)
 {
   print “<FONT face=’verdana’ color=#FF33CC><H3> El plantel es: </FONT></H3> Prepa 2”;
 }
if ($plantel==3)
 {
   print “<FONT face=’verdana’ color=#FF33CC><H3> El plantel es: </FONT></H3> Prepa 3”;
 }
if ($plantel==4)
 {
   print “<FONT face=’verdana’ color=#FF33CC><H3> El plantel es: </FONT></H3> Prepa ”;
 }
.
.
```

```
if ($asig1==1)
  {
    print "<FONT face='verdana' color=#FF33CC><H3> Windows XP</FONT></H3>";}
.
.
.
?>
</FORM>
</BODY>
</HTML>
```

En la Figura 5.2 muestra el formulario, diseñado como una página Web con HTML, pero ahora ya trabaja con PHP para enviar y recibir datos:

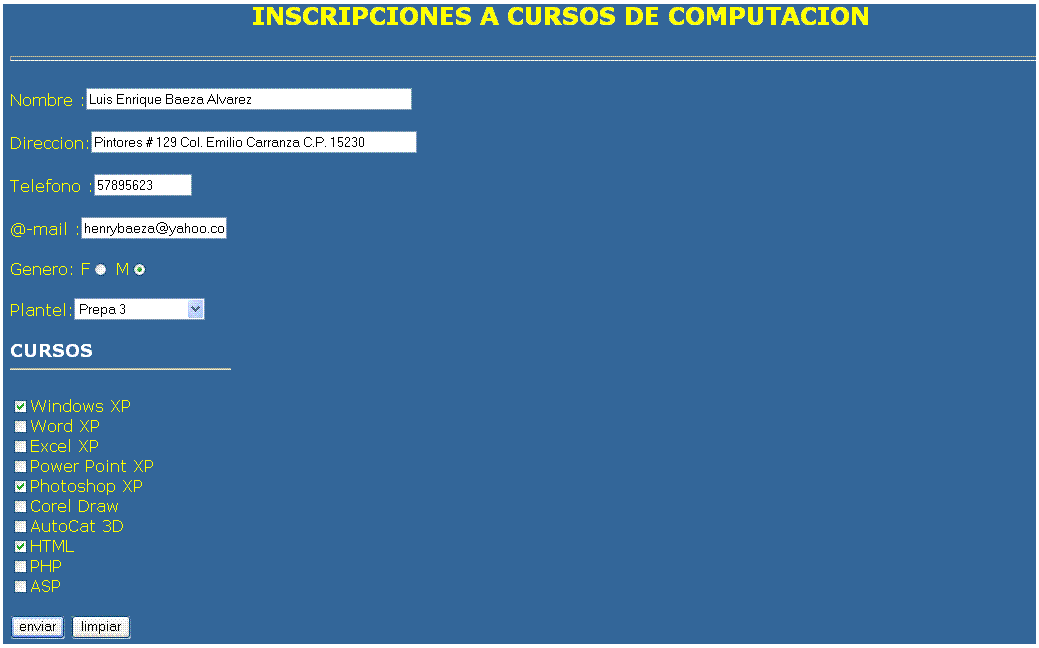


Figura 5.2. Formulario de Inscripciones.

Al pulsar el botón Enviar el contenido de cuadro de texto es enviado a la página que indicamos en el atributo ACTION de la etiqueta FORM, que en este ejemplo fue en Incripciones.php.

5.6.1. Funciones de acceso a ficheros

Posiblemente durante la tarea de programar se tenga la necesidad de obtener datos de un fichero, o bien, de crear uno. PHP provee de una extensa gama de funciones de acceso a ficheros.

En esta sección sólo se ven las funciones básicas, abrir ***(fopen)***, cerrar ***(fclose)***, leer ***(fgets)*** y escribir ***(fputs)***. Estas cuatro solventaran la mayoría de los problemas que se presenten con respecto al acceso a ficheros.

- **fopen (archivo, modo)**

Con esta función se abre un fichero, bien sea local o una dirección de internet (http:// o ftp://).

La función **fopen** devuelve un valor numérico (indicador de archivo) de tipo **integer** servira para hacer referencia al archivo abierto.

Con fopen se puede abrir un archivo de los siguientes modos:

r solo lectura

r+ lectura y escritura

w solo escritura. Sino existe el archivo lo crea, si ya existe lo machaca.

w+ lectura y escritura. Sino existe el archivo lo crea, si ya existe lo machaca.

a solo lectura. Sino existe el archivo lo crea, si ya existe empieza a escribir al final del archivo.

a+ lectura y escritura. Sino existe el archivo lo crea, si ya existe empieza a escribir al final del archivo.

Los modos **r**, **r+** , **w** , **w+** colocan el puntero de lectura/escritura a principio del fichero, los modos **a** , **a+** lo colocan al final.

- **fgets (indicador_archivo, longitud)**

La función *fgets* devuelve una cadena con la longitud especifica del fichero al que apunta el indicador de archivo.

La función *feof* devuelve TRUE si puntero de lectura/escritura se encuentra al final del fichero, y FALSE en caso contrario.

- **fputs (indicador_archivo, cadena)**

La función *fputs* escribe una cadena en el fichero indicado. Para escribir en un archivo este debe haber sido previamente abierto. La función *fputs* devuelve TRUE si se ha escrito con éxito, en caso contrario devuelve FALSE.

- **fclose (indicador_archivo)**

Con esta función se cierra el fichero que nos marca el indicador de archivo, devuelve TRUE si el fichero se cierra correctamente y FALSE sino se ha podido cerrar.

- **file_exists (fichero)**

Esta función devuelve TRUE si el archivo especificado existe, y FALSE en caso contrario.

- **copy (origen, destino)**

La función *copy*, copia un fichero de un lugar (origen) a otro (destino), devuelve TRUE si la copia a tenido éxito y FALSE en caso contrario.

5.7. Software para aplicaciones PHP

Para poder trabajar con PHP se deben de tomar ciertas condiciones necesarias en cuanto a software que se requiere instalar. Como todo lenguaje de lado servidor, PHP, requiere de la instalación de un servidor en la PC para poder trabajar localmente. Este modo de trabajo resulta a todas más práctico que colgar los archivos por FTP en el servidor y ejecutarlos desde Internet.

Así que antes de comenzar a crear programas en PHP, es necesario convertir un ordenador en un servidor. Esto se hace instalando uno de los varios servidores disponibles para el sistema operativo de la máquina, e introducir en nuestro servidor los archivos que le permitirán la comprensión del PHP.

La elección del programa servidor tendrá mucho que ver con el sistema operativo que se tenga corriendo en el ordenador.

Para los usuarios de Windows en general se recomienda instalar **Apache** como servidor Web lo cual puede resultar ventajoso con respecto al uso del PWS ya que PHP está principalmente diseñado para correr en este servidor. Esto quiere decir que, aunque en principio todo debería funcionar correctamente sobre ambos servidores, es posible que algún bug no corregido haga fallar uno de los scripts si se trabaja con un servidor cuyas actualizaciones son menos frecuentes y detalladas.

Apache ha sido especialmente pensado para plataformas Unix-Linux, aunque recientemente, con la Apache 2.0, han desarrollado una versión específica para Windows.

Se explicara como se instala el paquete **PHPTriad**. Estos paquetes son para facilitar la tarea de instalación del servidor ya que contienen todo lo necesario.

En este caso se enfocara la instalación sobre Windows XP, para obtener este paquete se puede conseguir las diferentes actualizaciones de la página oficial de Apache, una ves ya obtenido el paquete, se procederá a instalarlo.

1. Al inicio de la instalación se muestra una pantalla como se observa la Figura 5.3 que indica el proceso de instalación.

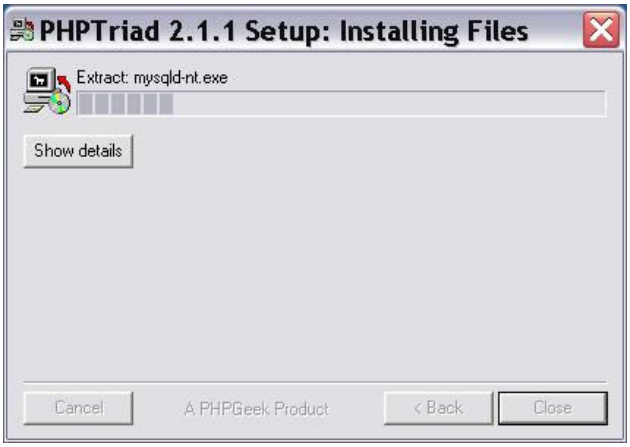


Figura 5.3.Proceso de Instalación.

A mitad de la instalación se aparecerá una ventana de Ms-Dos, esta ventana forma parte de la instalación.

2. Se debe iniciar los servicios. En [**Programas** -> PHPTriad], Desde ahí mismo se selecciona el submenú de ***Apache Console*** y a su ves, se le da a ***Start Apache*** como se muestra en la Figura 5.4. Deberá salir una ventana en Ms-dos, se minimiza únicamente (**no cerrar**). Seleccionamos el submenú ***Mysql*** y seleccionamos ***Mysql-D-NT***, tendremos un bonito semáforo donde también muestra un reloj. Si no sale el semáforo, que no entre el pánico. Se debe continuar con la instalación.

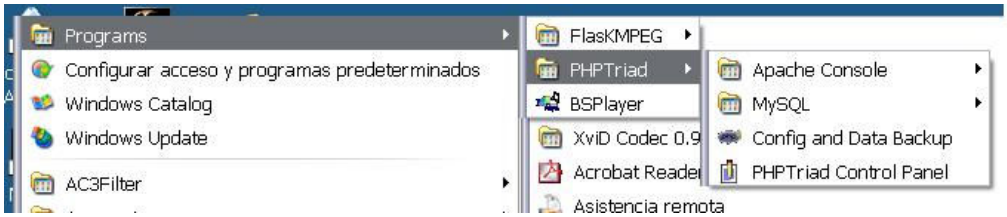


Figura 5.4. Localización de la Consola de Apache.

3. Para comprobar que funciona **Apache** correctamente se abre una ventana de navegador y se poner **http://localhost/** que debería funcionar siempre. Si aparece una ventana como la que muestra la figura 5.5, felicidades el servidor Web Apache esta andando. La Web que se muestra esta en la carpeta c:/apache/htdocs/, carpeta con la que hay que familiarizarse porque es donde van los documentos Web que puede mostrar Apache. También podemos meter Webs en carpetas dentro de htdocs como por ejemplo el MyPHPAdmin.

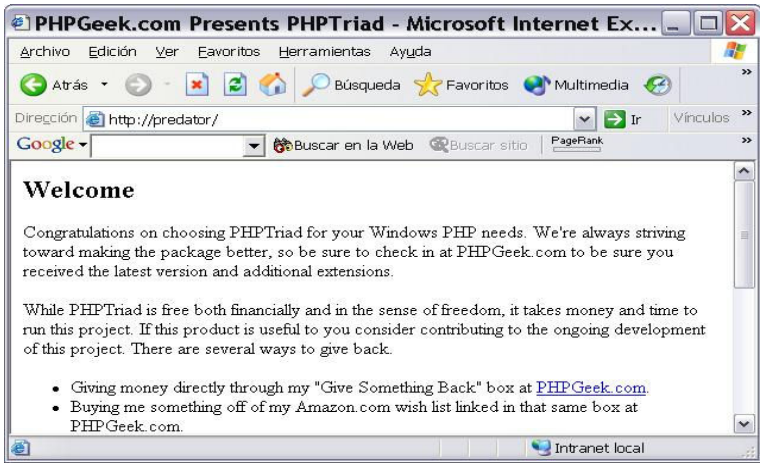


Figura 5.5. Página de bienvenida.

4. Ahora para comprobar si funciona el **MySQL** apunta con en el navegador a **http:///localhost/phpmyadmin/** si carga es que funciona. En la base de datos test se puede probar crear tablas y probar cosas para familiarizarse con el MyPHPAdmin que es muy útil.

Como se puede observar es muy sencilla la instalación de PHPTriad y su gran ventaja es que es un paquete que ya incluye todo lo que se requiere para poder comenzar a programar las aplicaciones que se necesitan, también se observa que a comparación de otras versiones de PHP no es necesario configurar manualmente el servidor Apache, sino que todos es automático, a comparación de otras versiones, en las cuales es necesario configurar el servidor Apache con PHP manualmente y es un poco mas complicado.

A lo largo de este capítulo se han fundamentado las bases para la programación con PHP y algunos ejemplos que pueden dar ideas para los futuros proyectos. Faltaría por comentar una de las mejores características de este lenguaje que es la sencilla conexión a bases de datos.

Por lo tanto se logra ver que PHP cuanta con muchas aplicaciones y ventajas como la capacidad de programar páginas Web dinámicas, habitualmente en combinación con el motor de base datos MySQL, aunque cuenta con soporte nativo para otros motores, incluyendo el estándar ODBC, lo que amplia en gran medida sus posibilidades de conexión. Programación en consola, al estilo de Perl, en Linux, Windows y Macintosh. Creación de aplicaciones gráficas independientes del navegado. Capacidad de acceder a la mayoría de las base de datos que se utilizan en la actualidad. Leer los datos desde diferentes fuentes, incluyendo datos que pueden meter los usuarios desde formas HTML y manipularlos de forma sencilla. Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados extensiones).Posee una muy buena documentación en su página oficial (www.php.net). Es libre, por lo que se presenta como una alternativa de fácil acceso para todos. Permite las técnicas de Programación Orientada a Objetos.

Conclusión

La finalidad de este diplomado como se dijo en un inicio esta enfocado principalmente en la actualización del personal docente en el ámbito de la programación de computadoras, lo cual es muy importante en cuanto a desarrollar un perfil del docente dentro de este contexto.

Ya que se comparte la preocupación por mejorar la calidad de la educación con equidad para responder oportunamente a las exigencias de nuestra sociedad.

El avance de la tecnología de la información propicia un cambio en el paradigma de la producción y divulgación del conocimiento; y en esto, el sistema educativo y las instituciones, que tienen el compromiso de asegurar el acceso al mismo se deben replantear cuáles son las competencias exigibles.

Para mejorar la calidad de la educación es necesario producir significativas transformaciones en el sistema educativo, profesionalizar la acción de las instituciones educativas y la función del docente.

El docente es el actor principal en el proceso de mejoramiento de la calidad educativa pues es el nexo en los procesos de aprendizaje de los alumnos y las modificaciones en la organización institucional. Las reformas educativas se traducen en las escuelas y llegan al aula por medio del docente.

Actualmente es importante que la sociedad cuente con maestros y profesores eficaces y eficientes para poner en práctica distintos y adecuados recursos y en las ocasiones oportunas, con el fin de acceder a mejores logros educativos.

De aquí la importancias y la necesidad de contar con este tipo de cursos de actualización docente enfocados a una área de estudios que en este caso el diplomado fue dedicado principalmente en cómputo y el aprendizaje de algunos lenguajes de programación, logrando resultados muy favorables, pero a su ves despertando un gran entusiasmo en seguir actualizándose, ya que el cómputo es una área de constante crecimiento en cuanto a tecnología de hardware, como en tecnología de software y para lograr resultados actuales y favorables para las futuras generaciones.

Se ha llegado a la conclusión que los cursos de actualización es una muy buena herramienta que hoy en día ha ido creciendo constantemente dentro de las instituciones ya que se ve la necesidad que se tiene por crecer, mejorar y sobre todo sobresalir como docente en diferentes áreas de estudio y contar con las herramientas de pedagógicas y didácticas de la enseñanza.

El desarrollo del diplomado se llevo a cabo a través de diferentes módulos que mostraron de forma gradual la solución de un problema dentro del contexto computacional y pasando por sus diferentes etapas de análisis hasta llegar a una posible solución por medio de la programación.

Se comenzó por estudiar lo que es lenguaje Pascal, ya que se basa en el paradigma de la programación estructurada, la cual es una teoría de programación que consiste en construir programas de fácil comprensión. Por lo mismo fue una excelente herramienta para introducirse en el tema principal logrando una mejor comprensión. Posteriormente se fueron desarrollando los demás módulos, los cueles ya requerían de conocimientos básicos.

Este diplomado cumplió con las expectativas ya que partió de lo básico a lo complejo como lo fueron los otros tipos de programación como la orientada a objetos y las basadas en aplicaciones Web, es decir el curso se llevo con cierto seguimiento entre sus módulos, y así posteriormente se podría pensar en algún diplomado que continué con los conocimientos adquiridos para seguir avanzando en la actualización docente.

Por otro lado un aspecto importante que también cubrió el curso y sobre todo haciendo énfasis en la actualización de docentes es la parte didáctica o de metodología de programación, es decir la parte pedagógica para lograr un buen aprendizaje, cubriendo los aspectos mas importantes para poder brindar un panorama general de lo que es la metodología y didáctica y la importancia de su aplicación al impartir clases.

Uno de los factores por los cuales se desarrolla este tipo de proyectos, es pretender facilitar el proceso de enseñanza-aprendizaje, para lograr incrementar el nivel de calidad académica en el profesor. Ya que como anteriormente se mencionó, la actualización docente significa poner al día las prácticas y métodos de docencia y enseñanza juntos, aplicar una buena metodología de la programación para lograr que los alumnos de nivel medio superior comiencen a interesarse en estos temas y aplicarlos como una herramienta potente para el desarrollo de su profesión.

En algo muy general se puede mencionar que el aspecto intelectual se reduce a saber realizar el éxito educativo, y para ello, es necesario saber concebirlo, prepararlo, organizarlo, ejecutarlo y explotarlo.

El éxito educativo es el fin de la práctica, en su doble sentido, pero a su vez, fuente motivadora de perfeccionamiento y actualización del docente en actividad. En este sentido, hemos generado un auténtico círculo virtuoso, que supone un constante mejoramiento de la calidad educativa.

Ahora, si podemos ver la aplicación del diplomado en el ámbito empresarial o en cuestión de desarrollo u aplicación en alguna institución, también es muy amplio el panorama en el cual se pueden desempeñar la construcción de programas institucionales, programas con aplicaciones docentes, etc.

Con ayuda del conocimiento que se obtuvo a través del curso y con la aplicación de los diferentes leguajes de programación se logró desarrollar material didáctico como apoyo a la docencia por medio de el lenguaje Java y sus aplicaciones con applets algunos profesores de matemáticas y de física desarrollaron pequeños programas para enseñar las diferentes teorías con las que cuentan estas asignaturas logrando resultados satisfactorios con una amplia comprensión por parte de los alumnos. Por otro lado, en el área de cómputo, se desarrollo un sistema por medio de una pagina Web desarrollada con HTML y aplicaciones PHP con la que se logró crear una base de datos para las inscripciones de los alumnos de las diferentes opciones técnicas del plantel y así como estos ejemplos se pueden mencionar muchos más que se han credo con finalidades diferentes para apoyo del mismo plantel o la institución docente.

En conclusión no cabe más que mencionar que la finalidad del curso o diplomado se ha logrado hasta el momento y no dejar de señalar la gran necesidad de seguir con este tipo de programas de actualización docente para lograr un mejor crecimiento en el profesorado y principalmente en el alumnado.

BIBLIOGRAFÍA

Baase, Sara. *Algoritmos computacionales: introducción al análisis y diseño.* 3ª Ed., México, Ed. Pearson Educación,2002, 686 pp.

Bells, Miriam. Técnicas didácticas de capacitación. México, Ed. Lucas Morea, 32 pp.

Charte Francisco. *Delphi 5.* Madrid. Ed. Anaya Multimedia, 1999, 352 pp.

Decaer, Rick. *Programación con Java.* 2ª Ed., México, Ed. Hamilton Collage, 2001, 606 pp.

Dirk, Luis. *Delphi 5.* Barcelon, Ed. Marcombo, 2000, 856 pp.

E. Hernandez. *C++ Estandar*, México, Ed. ITP Paraninfo, 2001, 129 pp.

Fabrega, Pedro Pablo. *PHP 4.* México, Ed. Prentice Hall, 2000, 341 pp.

Feria Victoria Angélica. *Didáctica de la Programación.* México, 2003, 42 pp.

Froufe Quintas, Agustin. *Java 2: Manual de Usuario y tutorial.* 2ª Ed. Madrid, Ed. Ra-Ma, 2000, 682 pp.

García de Jalón, Javier. *Aprenda Java como si estuviera en primero*. San Sebastián, 87 pp.

G. Booch. *El lenguaje unificado de modelado.* Ed. Addison Wesley, 1999, 165 pp.

I. Jacobson. *El proceso unificado de desarrollo.* Ed. Addison Wesley, 2000, 132 pp

Mendoza González, Omar. *Programación en Java: Introducción a Java.* México, 2003, 97 pp.

Peñaloza Romero Ernesto. *Didáctica de la Programación.* México, 2003, 95 pp.

Pérez Medel Marcelo. *Lenguaje de Programación Pascal.* México, 2003, 22 pp.

Presuman, Roger S. *Ingeniería del software: un enfoque práctico.* 4ª Ed., México, Ed. Mc Graw-Hill, 1998, 581 pp.

Ramírez Montalvo Ernesto. *Solución de Problemas y algoritmos*. México, 2003, 15 pp.

Reisdorph, Kent. *Aprendiendo Borland Delphi 4.* México, Ed. Prentice Hall, 1999, 944 pp.

Rodríguez, José Ignacio. *Aprenda Java como si estuviera en primero.* México, Ed. San Sebastián T, 1999, 159 pp.

Sanchi Llorca Francisco J. *Programación con el lenguaje pascal.* 2ª Ed., Madrid, Ed. Paraninfo, 1991, 362 pp.

Otras fuentes

Alvarez, Miguel Angel. *Manual de PHP.* [en linea] ed. 1.0. Mexico, 2000 [citado 25 de octubre 2005]. Disponible en Word Wide Web: http://desarrolloweb.com.

M.C. Castro, Jesús Antonio. Diseño estructurado de algoritmos. [en linea], M.C. Ayala R.,Mario. La Paz, 1999 [citado 17 de noviembre 2005], actualización mensual. Disponible en Word Wide Web: http://www.itlp.edu.mx/publica/algoritmos.html.