UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

ASIGNATURA DE PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA

AUTOR LUIS ARENAS HERNÁNDEZ

PRÓLOGO

Un apoyo que tiene el proceso enseñanza-aprendizaje, es precisamente el material didáctico, ya sea a través de apuntes, cuaderno de ejercicios, libros o aplicaciones de computadora sobre todo si se trata de un área tan dinámica como lo es la de la Informática.

Estas notas pueden ser de apoyo para la materia y se pretenden mostrar algunos ejemplos y conceptos de la programación orientada a objetos, enfocados a un lenguaje de programación de este tipo, que es JAVA.

Este trabajo, se desarrolla siguiendo dos ideas principales: la teoría o conceptos sobre lo que es la Programación Orientada a Objetos (POO) y lo que es en si su utilización en la programación en JAVA, para lo cual se divide en las siguientes partes:

I.- Introducción.
II.- Conceptos de la programación orientada a objetos en JAVA.
III.- Estructuras del Lenguaje.
IV.- Programación de interfases gráficas.
V.- Programación de Applets.
VI.- Programación Cliente/Servidor.
VII.- Bibliografía.
VIII.-Anexos.

Nota: En el *Anexo A,* se encuentra el programa detallado de esta asignatura.

I- INTRODUCCIÓN

La velocidad con que avanza en los últimos años la tecnología del Hardware de las computadoras, es muy grande, con lo que se ha logrado tener computadoras más poderosas, baratas y compactas. Pero el Software no ha tenido el mismo comportamiento, ya que al desarrollar aplicaciones, es frecuente que se excedan los tiempos de entrega así como los costos de los sistemas de información (tanto de desarrollo como de mantenimiento), además de ser poco flexibles.

Se han creado diferentes herramientas de ayuda al desarrollo, para lograr aumentar la productividad en el Software como son:

- Técnicas como las del Diseño Estructurado y el desarrollo descendente (top-down)
- Herramientas de Ingeniería de Software asistida por computadora (ordenador) conocida como CASE
- Desarrollo de lenguajes de programación más poderosos como los lenguajes de cuarta generación (4GL) y los orientados a objetos (POO).

- Diversas herramientas como gestión de proyectos, gestión de la configuración, ayuda en las pruebas, bibliotecas de clases de objetos, entre otras.

En donde se va hacer más énfasis es en los lenguajes de programación, sobre todo en los que es donde se tiene la Programación Orientada a Objetos (POO) y es donde se tienen lenguajes como JAVA, que es el que se va a utilizar en esta asignatura.
Un lenguaje en términos generales, se puede entender como "... los sistemas de símbolos y signos convencionales que son aceptados y usados individual y socialmente, con el fin de comunicar o expresar sentimientos, ideas, conocimientos , etc., por ejemplo, el lenguaje natural o articulado, el corporal, el artificial o formal, los sistemas de señalamiento, el arte, entre muchos otros tipos..." [1]
Existen también los lenguajes artificiales, que entre sus características, tenemos el que no es ambiguo y es universal, entre los que se encuentran los de las Matemáticas y los de Programación.
En los lenguajes de programación (conjunto de sintaxis y reglas semánticas con el fin de comunicar ideas sobre algoritmos entre las computadoras y las personas), existen diferentes clasificaciones una de ellas es la que a continuación se muestra:

- *Programación Imperativa,* donde el programa es una serie de pasos, realizando cada uno de ellos un cálculo (como ejemplos están Cobol, Fortran entre otros).
- *Programación Funcional,* el programa es un conjunto de funciones matemáticas que se combinan (como Lisp, Scheme, etc.).
- *Programación Lógica (también conocida como Declarativa),* aquí el programa es una colección de declaraciones lógicas (ejemplo Prolog).
- *Programación Concurrente,* la programación consiste en un grupo de procesos corporativos, que llegan a compartir información ocasionalmente entre ellos (ejemplos LINDA y Fortran de alto rendimiento HPF 1995)
- *Programación guiada por eventos,* el programa consiste en un ciclo continuo y que va a responder a los eventos generados aleatoriamente (orden no predecible), ya que dichos eventos son originados a partir de acciones del usuario en la pantalla (ejemplos JAVA y Visual Basic)
- *Programación Orientada a Objetos (POO),* el programa está compuesto por varios objetos que interactúan entre ellos a través de mensajes, los cuales hacen que cambien su estado (ejemplos C++, Eiffel y JAVA)

I.1- El paradigma orientado a objetos
Se tratará definir ¿qué son los paradigmas?, existen diferentes definiciones como que son las que a continuación se mencionan:

[1] Cuairán Ruidíaz María y Amelia Guadalupe Fiel Rivera, *Notas del curso de Orientación para la elaboración de textos didácticos de Ingeniería*. 2ª edición, México, Facultad de Ingeniería UNAM, 2004, págs. 7 y 8

- "...un conjunto de conocimientos y creencias que forman una visión del mundo (cosmovisión)..."[2].
- "...Un paradigma es una forma de representar y manipular el conocimiento. Representa un enfoque particular o filosofía para la construcción del software..."[3]
- "...Un modelo, ejemplo o molde..."[4]

"...La **Programación Orientada a Objetos (POO)** es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan *estado* (es decir, datos) y *comportamiento* (esto es, procedimientos o *métodos*)..."[5].

La programación orientada a objetos es un programa con un conjunto de objetos, que se comunican entre ellos para realizar tareas y que es un modelo que representa un subconjunto del mundo real, tal fielmente como sea posible, de modo fácil y natural, donde los objetos van a tener características (atributos) y comportamientos (métodos). Que a diferencia de los lenguajes procedurales, en donde los datos y los procedimientos se encuentran separados y sin relación alguna.

Los lenguajes procedurales, utilizan funciones y después les pasan datos, en tanto que los lenguajes orientados a objetos definen objetos y después envían mensajes a los objetos diciendo que realicen alguno de los métodos especificados para el objeto.

Entre las ventajas de la programación orientada a objetos es que los métodos están pensados para hacer programas y módulos más fáciles de escribir, mantener y reutilizar, así como que sean modulares y reutilizables parte de los códigos de estos programas.

A continuación se describirán sus mecanismos básicos de la POO como son: *objetos, mensajes, métodos y clases,* los cuales se describen brevemente a continuación.

Objetos.

Los Objetos, se pueden definir como las unidades básicas de construcción, para la conceptualización, diseño o programación, esto es que son instancias agrupadas en clases con características en común y que son los atributos y procedimientos, conocidos como *operaciones o métodos*.

También se puede decir que un *objeto* es una abstracción encapsulada genérica de datos y los procedimientos para manipularlos, también se puede decir que es una cosa o

[2] http://www.monografias.com/trabajos16/paradigmas/paradigmas.shtml#queson

[3] http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n

[4] FREEDMAN ALAN, *Diccionario de Computación*, (5ª edición.), MÉXICO, McGraw-Hill, 1993, pág. 581.

[5] http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos

entidad, que tiene atributos (*propiedades)* y de formas de operar sobre ellos que se conocen como *métodos*.
En forma mas simple se puede decir que un *objeto* es un ente que tiene características y comportamiento.
Los objetos pueden modelar diferentes cosas como; dispositivos, roles, organizaciones, sucesos ,ventanas (de Windows), iconos, etc.

Organización de los Objetos
Los objetos forman siempre una organización jerárquica, existiendo varios tipos de jerarquía como los que a continuación se mencionan:

- *Simples.-* cuando su estructura es representada por medio de un árbol (estructura de datos).
- *Compleja.-* cualquier otra diferente a la de árbol.

Sea cual fuere la estructura se tienen en ella los siguientes tres niveles de objetos:

- *La raíz de la jerarquía.* Es un objeto único, esta en el nivel más alto de la estructura y se le conoce como objeto Padre, Raíz o Entidad.
- *Los objetos intermedios.* Son los que descienden directamente de la raíz y que a su vez tienen descendientes (tienen ascendencia y descendencia) y representan conjuntos de objetos, que pueden llegar a ser muy generales o muy especializados, de acuerdo a los requerimientos de la aplicación.
- *Los objetos terminales.* Son todos aquellos que tienen ascendencia, pero que no tienen descendencia.

Mensajes
En la programación orientada a objetos, los objetos descritos anteriormente se comunican a través de señales o mensajes, siendo estos mensajes los que hacen que los objetos respondan de diferentes maneras, por ejemplo un objeto en Windows como una ventana de alguna aplicación , puede cerrarse, maximizarse o restaurarse (métodos) de acuerdo al mensaje que le se enviado.
En otras palabras, un mensaje es una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos pudiendo o no llevar algunos parámetros.

Métodos
Es una acción que determina como debe de actuar un objeto cuando recibe un mensaje.
En analogía con un lenguaje procedural se le llamaría “función”.
Un *método* también puede enviar mensajes a otros objetos, para realizar una aación o para pedir información.

Clases, Superclase y Subclase
Es la generalización de un tipo especifico de objetos, esto se puede decir como el conjunto de características (atributos) y comportamientos de todos los objetos que componen a la clase.
Por ejemplo la clase plumón tiene todas las características (tamaño, color, grosor del punto, etc) y todos los métodos o acciones (pintar, marcar, subrayar, etc), que pueden tener todos los plumones existentes en la realidad.
Un plumón en especial como un marcador permanente para discos compactos (CDs) de color negro y punto fino, es un objeto (o instancia) de la clase plumón.

Se tienen tres tipos de clase que son:

- *Abstracta*.- Es muy general (ejem. Animal).
- *Común* .- Es intermedia (ejem. Mamíferos).
- *Final.*- Es muy especifica (ejem. GatoSiames).

La *herencia* (que se detallara más adelante), maneja una estructura jerárquica de clases o estructura de árbol (Estructura de Datos[6]), con lo que la POO todas las relaciones entre clase se ajustan a esta estructura. En esta estructura cada clase tiene una sola clase padre, la cual se conoce como *superclase* y la clase hija de una superclase se conoce como *subclase.*

Superclase
La *superclase,* se puede definir en términos sencillos como la clase padre de alguna clase especifica y puede tener cualquier número de subclases..

Subclase
La *subclase,* es la clase hija de alguna clase especifica y solo puede tener una superclase (en JAVA).

I.2- Principios fundamentales de la POO

Los principios fundamentales de la POO son: *abstracción, encapsulamiento, herencia y polimorfismo,* los cuales se describirán a continuación.

Abstracción
Es el proceso de representar entidades reales como elementos internos a un programa, la abstracción de los datos es tanto en los atributos, como en los métodos de los objetos. Es de hacer notar que por medio de la abstracción se puede tener una visión global del tema, por ejemplo en la clase PILA (Estructura de Datos, Ultimas Entradas Primeras Salidas LIFO), se pueden definir objetos de este tipo, tomando únicamente las propiedades LIFO (del inglés Last Input First Output) de las PILAS, algunos atributos como lleno, vacío, el tamaño y las operaciones o métodos que se pueden realizar sobre estos objetos como PUSH (meter un elemento) o POP (retirar un elemento) y no es necesario que el programador conozca o sea un experto en la Estructura de Datos con la que se implementa la PILA (como por ejemplo con una organización contigua o simplemente ligada, ni de los algoritmos que realizan el PUSH y el POP).

Encapsulamiento
Cada objeto está aislado del exterior, esta característica permite verlo como una caja negra, que contiene toda la información relacionada con ese objeto. Este aislamiento

[6] EUÁN AVILA JORGE IVAN Y CORDERO BORBOA LUIS GONZAGA**.**, *Estructuras de datos,* (1ª reimpresión.), MÉXICO, LIMUSA, tomada de la primera edición de la UNAM (FACULTAD DE INGENIERÍA), 1989, págs 89- 110.

protege a los datos asociados a un objeto para que no se puedan modificar por quien no tenga derecho a acceder a ellos.
Permite manejar a los objetos como unidades básicas, dejando oculta su estructura interna.

Herencia.
Las clases se encuentran relacionan entre sí, formando una jerarquía de clasificación. La *herencia* es el medio para compartir en forma automática los métodos y los datos entre las clases y subclases de los objetos. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.
Existe la *herencia Simple y Múltiple,* si un objeto pertenece a más de una clase, se llama *herencia Múltiple*, y no está soportada por algunos lenguajes como Java, en tanto que la *herencia Simple,* es cuando un objeto pertenece a una sola clase.

Como se ejemplifica en la siguiente figura 3.1:

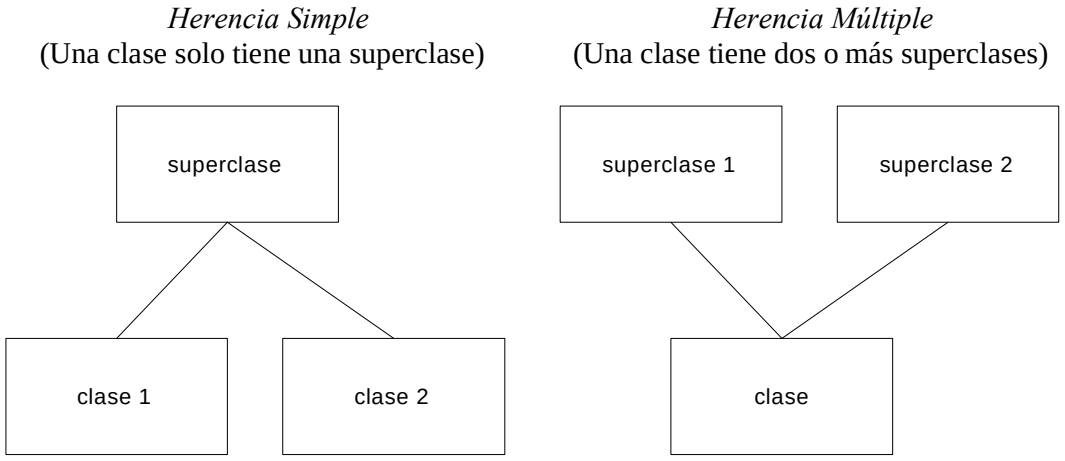

FIGURA 1.1 ***Herencia Simple y Herencia Múltiple.***

Polimorfismo
Esta característica facilita la implementación de varias formas de un mismo método, con lo cual se puede acceder a varios métodos distintos, que tienen el mismo nombre.
Existen dos tipos principales de *polimorfismo*, que son:

- Por *reemplazo.*- dos o más clase diferentes con el mismo nombre del método, pero haciéndolo de forma diferente.
- Por *sobrecarga.*- es el mismo nombre de método ocupado varias veces, ejecutándolo de diferente forma y diferenciándose solamente por el argumento o parámetro.

A continuación se ejemplifican en la figura 1.2 estos dos tipos de *polimorfismo.*

Polimorfismo por reemplazo *Polimorfismo por sobrecarga*

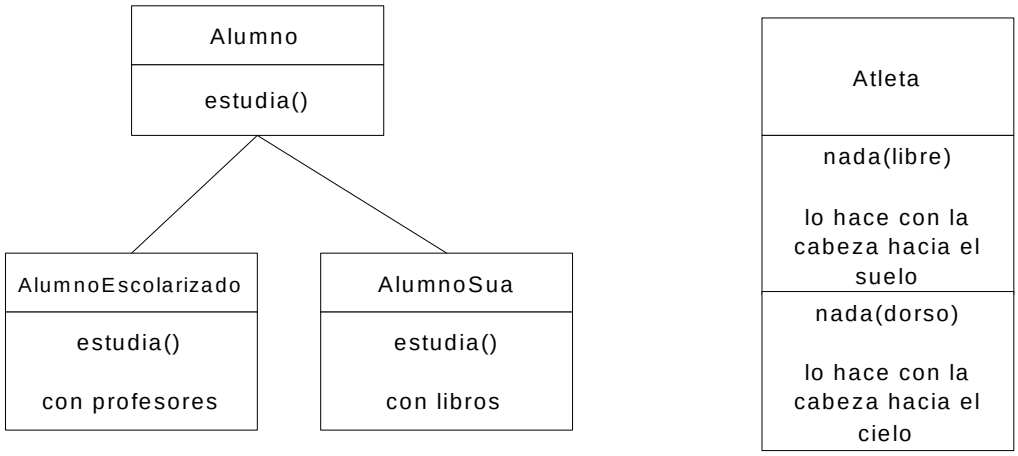

FIGURA 1.2 .*Polimorfismo por reemplazo y por sobrecarga.*

I.3.- Análisis y diseño orientado a objetos

Se han desarrollado metodologías, que tienen como una de sus funciones el lograr una mayor productividad en el desarrollo de los sistemas de información. Existen diferentes metodologías las cuales tienen diferentes fases y son precisamente dos de ellas en las que se enfatizará y son el Análisis y el Diseño.

Tal vez no sea tan importante ver cual es la mejor metodología, si no el conocer alguna y aplicarla, en general en la fase del Análisis se identifica el ¿**qué** se desea hacer?, es un Análisis detallado de los requerimientos del proyecto que se va a realizar, en el caso de la POO es la abstracción resumida y precisa de lo que debe de hacer el sistema deseado y en la del Diseño ¿**cómo** se va a hacer?.para cumplir con esos requerimientos, que características de rendimiento hay que optimizar.

En el caso de la POO, existen diferentes metodologías que consisten en construir un modelo (Representación de la realidad a través de diferentes variables) de un dominio de aplicación como:

- OMT que es la Técnica del Modelado de Objetos, el cual a grandes rasgos cuenta con las siguientes cuatro fases: *Análisis, Diseño del Sistema, Diseño de Objetos e Implementación.*[7]
- UML más recientemente, el cual es un Lenguaje Unificado de Modelado y es una representación gráfica que sirve para modelar sistemas orientados a objetos, ya que permite manejar los elementos descritos en los apartados anteriores (por ejemplo los mensajes entre objetos, sincronización , etc.).

  Entre sus principales características se encuentran: su flexibilidad, cuenta con muchos elementos gráficos.[8]

[7] RUMBAUGH JAMES, BLAH MICHAEL, PREMERLANI WILLIAMS, HEDÍ FREDERICK Y LORENSEN WILLIMAS. *Modelado y diseño orientado a objetos METODOLOGÍA OMT (*S/E), ESPAÑA, PRENTICE-HALL, España 1996, págs. 643.

[8] POOLEY PERDITA, *Utilización de UML en Ingeniería del Software con Objetos y Componentes* (S/E), ESPAÑA, MC-ADDISON WESLEY DE PEARSON EDUCACIÓN S.A., Madrid España 2003, págs. 291.

Al proceso del Análisis y Diseño orientado a objetos, se conoce bajos las siglas **ADDO.**

I.4.- Lenguajes orientados a objetos

Las técnicas, vistas anteriormente en las que se basa la programación orientada a objetos (como el encapsulamiento, abstracción , etc) ya eran conocidas, desde hace ya varios años, sin embargo no todos los lenguajes proporcionan todas las facilidades para escribir programas orientados a objetos. Existen discrepancias de cuales deben de ser estas facilidades y se pueden agrupar en las siguientes:

- Manejo de memoria automático, incorporándose el concepto del recoletor de basura, con lo que la memoria utilizada por objetos cuya utilidad ha terminado es liberada por mecanismos propios del lenguaje, sin intervención del programador.
- Abstracción de datos a través del lenguaje.
- Estructura modular en objetos, tomando como base sus estructuras de datos.
- Clases, Herencia y polimorfismo que puedan manipuladas a través del lenguaje.

Entre los lenguajes orientados a objetos destacan los siguientes:

- Smalltalk
- Objective-C
- C++
- Ada 95
- Java
- Ocaml
- Python
- Delphi
- Lexico (en castellano)
- C#
- Eiffel
- Ruby
- ActionScript
- Visual Basic
- PHP
- PowerBuilder
- Clarion
- Simula67

Existen lenguajes como C++ y otros lenguajes, como OOCOBOL, OOLISP, OOPROLOG y Object REXX, han sido creados añadiendo extensiones orientadas a objetos a un lenguaje de programación ya existentes.

Una nueva tendencia en la abstracción de paradigmas de programación es la Programación Orientada a Aspectos (POA), pero esta metodología aún se encuentra en proceso, aunque cada vez más y más investigadores e incluso proyectos comerciales en todo el mundo ya la empiezan a adoptar.[9] [10]

II CONCEPTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA

II.0 Introducción a JAVA y como instalarlo

Java es un lenguaje de programación de alto nivel que cuenta con las siguientes características[11]: [12]

- Sencillo
- Orientado a objetos.
- Distribuido y dinámico
- Interpretado
- Robusto
- Seguro
- Neutral desde el punto de vista de la estructura
- Portátil
- De alto rendimiento
- Multitarea
- Multiplataforma (Mac, Windows y Linux)
- Con administración de memoria y recolección de basura

JAVA es un lenguaje de programación que utiliza un compilador, para traducir del código fuente al código ejecutable, aunque también tiene la característica de ser un lenguaje interpretado. El compilador de JAVA genera a partir del código fuente de un programa un código intermedio llamado *bytecode,* el cual es independiente de la plataforma en que se trabaje y este *bytecode* solo se puede ejecutar en la Máquina Virtual de Java conocida como **JVM,** la cual es una estructura idealizada de JAVA y en la mayoría de los casos es utilizada para implementar software en vez de hardware.

---

[9] http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos

[10] JOYANES, Luis. Programación Orientada a Objetos, segunda edición, MC-GRAW HILL/INTERAMERICANA DE ESPAÑA, Madrid 1998, p. 37.

[11] TUCKER ALLEN Y NOONAN ROBERT, *Lenguajes de Programación principios y paradigmas*, (S/E), ESPAÑA, MC-GRAW HILL/INTERAMERICANA DE ESPAÑA, Madrid 2003, pág. 443.

[12] ZUKOWSKI JHON, *Programación en Java 2* (S/E), ESPAÑA, EDICIONES ANAYA MULTIMEDIA, Madrid 1999, pág. 40-44.

JAVA se esta haciendo muy popular por sus características dentro de los grupos de desarrolladores, sobre todo con el auge que ha tenido Internet en los últimos años , a pesar de ser un lenguaje relativamente joven (principios de los años noventas), por darse una idea de su rápida aceptación, en las conferencias de "JAVA Soft Java One" en San Francisco, desarrollada en Abril de 1996 trajo la atención de cinco mil usuarios, al año siguiente fueron diez mil y en la actualidad no se sabe con exactitud, pero a finales del siglo XX se estimaban redondeaba la cifra de cuatrocientos mil y siguen creciendo.

La plataforma JAVA se basa solamente en software que corre sobre de las plataformas basadas en hardware y tiene los siguientes dos componentes:

- La JVM.
- La interfaz de Programación de Aplicaciones de JAVA conocida como ***API JAVA.***

En la figura 2.1 se hace una representación de los componentes de la plataforma JAVA:

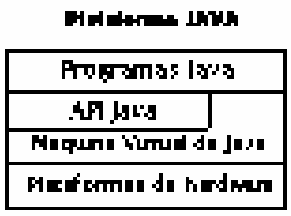

FIGURA 2.1 ***Plataforma JAVA***

Tipos de programas en JAVA
Los programas en JAVA tiene tres tipos formatos que son: *Applets, Aplicaciones y Servlet* y son los que a continuación se describen:

- *Applets.*- Los applets son pequeños programas que se incrusta en una página Web y por un Navegador Web compatible con Java para poder ejecutarse. Es frecuente que los *applets* se descargan junto con una página HTML desde un Servidor Web y se ejecutan en la máquina cliente. Un *applet* recibe la información directamente del explorador.
- *Aplicaciones.*- Las aplicaciones también son programas conocidos como standalone y que son ejecutados desde la línea de comandos del sistema operativo y son todos los programas que no caen en ninguna de las otras dos caterias (*applets o servlets).*
- *Servlets*.-Los servlets son programas especiales que se ejecutan en el servidor Web.

Compilación, obtención del JDK y ejecución de programas Java

El Kit de desarrollo de Java conocido como JDK (de sus siglas en Inglés), tiene los

elementos como son las herramientas y librerías necesarias para crear y ejecutar *applets* y *aplicaciones* en Java. JDK puede crear y mostrar aplicaciones gráficas.

Entre las librerías que tiene el JDK se encuentran las siguientes:[13]

- **javac**. Es el nombre del compilador de Java y convierte el código fuente escrito en Java (archivo con extensión java) a *bytecode* (archivo con extensión class).
- **java .** Es el interprete de JAVA, ejecuta el *bytecode* de los archivos con terminación punto class.
- **Appletviewer.**-Es el intérprete en JAVA que se encaraga de ejecutar la clase de los *applets* que estan dentro de los archivos HTML.
- **javadoc.-** Ayuda a documentar, ya que crea código HTML a partir del código fuente de JAVA y en los comentarios que contiene.
- **jdb.-** Es el analizador de JAVA, para poder recorrer línea a línea el programa, pudiendo analizar las variables y los puntos de interrupción.
- **javap.**- Es el desensamblador de JAVA y muestra las funciones a las que se pueden tener acceso y que se encuentran dentro de los archivos con extensión class.

Es de hacer notar que existen muchas más utilidades, las mencionadas, son las más comunes.

Para obtener el **JDK,** se puede hacer en gratuitamente en la pagina de SUN http://java.sun.com/ o http://javasoft.com/ .
Los requerimientos de memoria, varían de acuerdo a la versión y a las utilerias que se instalaran, aproximadamente 150 megas en disco duro.

Instalación del JDK para Windows 95 o 98
El JDK se encuentra en archivos que se auto extraen y es necesario indicar el directorio donde se colocarán los archivos y directorios del JDK.

Una vez que se ha instalado el JDK, una forma de empezarlo a utilizar es desde una ventana del sistema operativo DOS, ir al directorio donde se instaló el Kit de la siguiente forma:

cd j2sdk1.4.0_01\bind

suponiendo que el directorio y subdirectorio de instalación fue el j2sdk1.4.0_01\bind

Construcción de aplicaciones con el JDK

Para construir una aplicación, se requiere de utilizar algún editor de textos como puede ser Word o Bloc de Notas por ejemplo y lo que si es muy importantes es que se necesita salvar con la extensión **java** y ponerlo entre comillas, como se muestra a continuación y que se este en el directorio donde se instaló el **KIT** (por ejemplo el **j2sdk1.4.0_01\bind)**:

"HolaContaduria.java"

[13] Idem.

Otro aspecto importante, es que el nombre del archivo debe de coincidir con el nombre de la clase y la extensión será **java.**
Las convenciones que se siguen para el nombre de las clases es que, la primera letra sea mayúscula, seguida de letras minúsculas (ó números) y si lleva más de una palabra, la primera letra de cada palabra, también empezará con una letra mayúscula.

A continuación se muestra un ejemplo:

- Se edita el código fuente.

```
public class HolaContaduria
{
   public static void main(String args[])
   {
      System.out.print("Hola Contaduria");
   }
}
```

- Se guarda en un archivo con extensión **java** y entre comillas, para que sea un archivo de **texto**

  Guardar como y en nombre del archivo se pondrá
  "HolaContaduria.java"

- Se compilará el programa con el siguiente comando:

  javac HolaContaduria.java

  Si se llegan a tener errores de compilación, en este momento los desplegará y si no se tiene ningún error, entonces se generará, un archivo con extensión **class.**

- Se ejecutará el programa con:

  java HolaContaduria

  Y desplegará los resultados que arroje el programa.
  Es de hacer notar que al ejecutar el programa no se pone ninguna extensión (que de hecho sería la de **class).**
  Los archivos que se generaron fueron:

  HolaContaduria.java y HolaContaduria.class

A continuación se pasará a lo que es en si el manejo en JAVA de los conceptos mencionados de la POO en el puno UNO del temario.

II.1.- Clases de Objetos
Definición de clases

Cuando se define una clase se especifica como serán los objetos de dicha clase, esto quiere decir, de que variables y de que métodos estará constituida. Las clases proporciona una especie de plantilla o molde para los objetos, como podría ser el molde para hacer paletas de hielo, los ***objetos*** que se crean, son en sí las paletas de hielo, estas paletas van a tener como ***atributos (características),*** su color, tamaño, sabor, etc y como ***acciones o métodos (acciones),*** que se pueden congelar, derretir, etc.

Declaración de clases
Para declarar una clase, se realiza utilizando la palabra reservada ***class,*** seguida del nombre de la clase como se ejemplifica a continuación:

```
class nombreClase                    // Se declara la clase
{
        .
        .// Aquí se declaran las variables y los métodos
        .
}
```

La definición de la clase tiene dos partes que son:

- **La declaración de la clase** Indica el nombre de la clase precedido por la palabra clave (o palabra reservada) **class**.
- **El cuerpo de la clase** sigue a la declaración de la clase y está contenido entre los delimitadores **{ y }** y contiene las declaraciones de las variables de la clase, así como la declaración e implementación de los métodos que operan sobre dichas variables. También se le conoce como elementos miembros.

Declaración de variables de instancia

El estado de un objeto se representa por sus variables (conocidas como variables de instancia). Estas son declaradas dentro del cuerpo de la clase.
En JAVA a un ***objeto*** se le conoce como instancia de una clase y para crearlo se llama a una parte de líneas de código conocidas con el nombre ***constructor*** de una clase que tienen el mismo nombre de la clase.
Una vez que ya no se ocupa el objeto, se ejecuta el ***recolector de basura.*** Estos concepto de ***objeto, constructor*** y de ***recolector de basura,*** se abordarán más a detalle en el siguiente capítulo.

II.2.- Métodos y atributos de los objetos

Un objeto tiene atributos y métodos como se vio anteriormente, a continuación se describirán como se puede definir estos elementos en JAVA.

Métodos.

Los métodos interactúan con los mensajes, determinando cuales son los que un objeto puede recibir. Haciendo una analogía con la programación procedural, se podría comparar con las funciones o subrutinas.
Un método cuenta con las siguientes partes:

- Nombre del método.
- Valor que regresa.
- Argumentos opcionales.
- Modificadores en la declaración del método (como los de visibilidad)
- Cuerpo del método.

Nota: lo que se encuentra entre los picoparéntesis es opcional, en las sintaxis que se mostrarán en estas notas.

La sintaxis del método es:

```
<Modificadores>  valor_de_retorno  nombre_del_Metodo( <lista de argumentos > )
{

        // cuerpo del método


}
```

Recordando que los métodos, es lo que hace un objeto (es la acción) y cuando se llama a un método de un objeto es cuando se realiza el envío del mensaje al objeto. En **JAVA solo pueden ser creados como parte de una clase**.

Atributos de los objetos
Los atributos son las características del objeto (que tiene, de que consta), en JAVA se definen a través de tipos de datos o variables, los cuales se dividen en:

- *Tipos de Datos Primitivos (TDP).-* Solo se pueden manejar asignándoles algún valor y se dividen en boléanos (lógicos) y numéricos
- *Tipos de Datos por Referencia.-* Aparte de asignarlos, se pueden con él a través de una referencia, que haciendo una analogía con los lenguajes procedurales, equivaldría a los apuntadores.

En el siguiente capitulo, se profundizará un poco más en cuanto a los diferentes tipos de datos, viendo una clasificación más detallada.

Se toma la convención de que al utilizar estos atributos en JAVA, se utilicen siempre con letras minúsculas y solo si se trata de más de una palabra, entonces será con mayúscula cada primera letra de cada palabra adicional y siempre juntas, sin espacios en blanco, por ejemplo:

colorDelPelo

II.3.- Herencia

Una de las principales características de JAVA es la reutilización del código, para la reutilización del código, se crean nuevas clases, pero utilizando las clases ya existentes y la estructura jerárquica de clases o de árbol que se mencionó anteriormente (en donde todas las relaciones entre clases se ajustan a dicho árbol). Para la reutilización del código se utilizan los siguientes dos métodos más frecuentemente:

- *Composición.*- Donde se crean objetos a partir de clases existentes, la nueva clase se compone de objetos de clases ya existentes.
- *Herencia.*- Es como en la vida real, alguien hereda, adquieres algún bien o característica genética de su padre, se puede decir: esta persona heredó hasta ola forma de caminar de su padre, en JAVA equivaldría a decir: esta clase es como su clase padre, ya que conserva los atributos y los métodos de la clase padre, pudiendo "alterar" (por medio de los modificadores de acceso), ya sea para quitar o agregar algunos de ellos en forma individual para esa clase en especifico. Para manejarlo en JAVA se utiliza la palabra reservada ***extends.***

En esta estructura de árbol solo se tiene una clase padre (recordando que JAVA únicamente maneja herencia simple), la cual se conoce como *superclase* y en JAVA se invoca con la palabra reservada ***super*** y l clase hija de una superclase se llama subclase.

II.4- Polimorfismo

Como se vio en el capitulo anterior existen diferentes tipos (sobrecarga y reemplazo), solo se podrían agregar los siguientes comentarios:

Un objeto solamente tiene la forma que se le asigna cuando es construido, pero se puede hacer referencia a él en forma polimórfica, ya que se puede referir a objetos de diferentes clases y debe de existir una relación de herencias entre clases.

El Polimorfismo permite separar el ***que*** del ***como*** y permite distinguir entre tipos de objetos que son parecidos, esta diferencia se muestra través del comportamiento de los métodos que heredan de la clase padre.

Las funciones (métodos) polimórficas, son a las que no les interesa el tipo de variables que reciben, como es el caso del print o write en lenguajes como BASIC o PASCAL, que pueden desplegar, cualquier tipo de datos (entero, flota, string, etc.)

En el siguiente capitulo se mostrará un ejemplo donde estarán los conceptos vistos en este capitulo, para ya tener más bases en cuanto a la ***sintaxis*** del lenguaje.

III. ESTRUCTURA DEL LENGUAJE

La sintaxis (lo equivalente a la gramática en un lenguaje natural) de un programa escrito en JAVA es casi, casi igual a uno escrito en el lenguaje de programación C++. Generalmente las instrucciones terminan con un punto y coma ; con las excepciones que maneja también en C/C++.

III.1.- Tipos de datos

Los tipos de datos en JAVA son los siguientes tres:

- *Primitivos.*- Son unidades de información tales como caracteres, números o valores boléanos (lógicos).
- *Clases del sistema.*- No son clases y no tienen métodos propios.
- *Clases definidas por el usuario.*- Como su nombre lo indica son hechas por el usuario.

Los tipos primitivos en JAVA son:

- *bolean.*- Los cuales pueden tener valores de falso o verdadero (trae/false).
- *char.*-Es un carácter de 16 bits.
- *byte, short.*- Cuenta con 8, 16,32 y 64 bits
- *int, long.*- Tiene valores entero y enteros largos.
- *Flota, double.*- Son números de punto flotante (notación científica) de 32 y 64 bits.

Dentro de la sintaxis de JAVA se encuentran los siguientes elementos:

- *Comentarios.*
- *Declaraciones.*
- *Bloques de código.*
- *Estructura de archivos fuentes.*
- *Identificadores.*
- *Palabras claves*
- *Literales.*
- *Expresiones y Operadores*

Se describirán a continuación los más importantes.

Comentarios

Son las anotaciones que se utilizan por el programador, con el fin de documentar el código fuente, pero que no forma parte de ninguna instrucción. En JAVA se pueden utilizar los dos sistemas que utiliza C/C++ que son el de /* */ y //, además se puede agregar un tercero, para cuando se genera el archivo de documentación, que se explica enseguida,

Comentarios en una o varias líneas.

Es cuando se utilizan los símbolos /* y */

Ejemplo

```
/*
        Ejemplo de comentario
```

```
que puede utilizar
varias
líneas

*/
```

Comentarios de una sola línea.
Se utiliza la doble diagonal //, el comentario se inicia a partir de la doble diagonal y continua solo hasta el fin de la línea.

```
// Ejemplo del comentario en una sola línea
```

Comentarios para documentación
Recordando que entre las librerías que tiene el JDK, se encuentra la de ***javadoc***, que tiene como función el generar un archivo de documentación, se le pueden agregar comentarios dentro de ese archivo, cuando en el código fuente se utiliza el comentario del asterisco, pero agragandole otro.

Ejemplo

```
/**
Este comentario va a ser utilizado por la
Herramienta javadoc
y se agregara en
en ese archivo
*/
```

Declaraciones
Es alguna línea del programa.

Ejemplo

```
a = b + c;
```

Bloques de código
Es un grupo de declaraciones (líneas de código, o sentencias) que tienen alguna funcionalidad. Va encerrado entre las llaves **{** y **}**, se puede poner en cualquier parte donde se coloca una sentencia individual.

Ejemplo

```
public static void main(String args[])
{                                       /*   Aquí empieza el bloque
  Vuelo avion=new Vuelo();
  avion.altitud=25;
  avion.altura=50;
  avion.velocidad=1000;
  avion.latitud=500;
}                                           Aquí termina el bloque */
```

En el punto **III.2,** se profundizará en lo que son los bloques y las sentencias.

Estructura de archivos fuente
Los archivos fuente, tiene los siguientes tres tipos de declaraciones, que se encuentran fuera del bloque del código:

- *packages.*- Es la definición del paquete a que pertenecen las clases del archivo.
- *import.*- Da la referencia a una clase ya establecida y solo se utiliza el nombre de la clase.
- *class.*- Es donde se define la clase principal.

Ejemplo

```
package com.sybex.examples;              // Se define un package
import java.io.PrintWriter;              // Se define un import
public class PointTest {                 // Se define una clase
 public static void main(String args[])
{

                    // cuerpo del programa

 }
```

Identificadores
Es el nombre bajo el cual se pueden manipular las variables y son una secuencia de caracteres, los cuales deben de comenzar con una letra y ser seguido de letras y/o números, signos especiales como "_" y "$", pudiendo ser las letras mayúsculas y/o minúsculas.
No existe límite, en cuanto al número de caracteres en el nombre del identificador, pero se recomienda que no sea muy largo, ya que se puede prestar a cometer errores y se sugiere que sea un nombre representativo de lo que contiene o maneja.
Existen también variables de instancia (de objetos) que se detallaran en una sección más adelante.

Ejemplos

Color a AvionComercial Avion_Militar

Palabras clave
Son palabras o identificadores que tienen una función especial en el compilador de JAVA y que no pueden utilizarse como nombres de atributos y/o de métodos, en la tabla 3.1 se muestran las que utiliza JAVA.

abstract	default	if	package	synchronized
boolean	do	implements	private	this
break	double	import	protected	throw
byte	else	instanceof	public	throws
case	extends	int	return	transient
catch	final	interface	short	try
char	finally	long	static	void
class	float	native	strictfp**	volatile
const*	for	new	super	while
continue	goto*	null	switch	widefp

TABLA 3.1 ***Palabras claves en JAVA***

Donde:
* Son palabras claves de JAVA que no se usan mucho actualmente, pero que versiones anteriores si las usaban.
** Son palabras clave utilizada en la versión de JAVA 2.
true y false vistos como valores de tipos de datos primitivos, tampoco pueden ser usados como identificadores.

Literales
Son valores constantes, constituidos por un conjunto de caracteres y que se escriben de forma independiente, esto es en otras palabras se dice que un identificador es el símbolo de un valor y la literal es el valor mismo. Son ejemplos de literales los números (en cualquiera de sus tipos), los caracteres y la cadena de caracteres

Ejemplo

```
Identificador     Literal
NumDeVuelo = 747
```

Existen diferentes tipos de literales como las que a continuación se mencionan:

- ***Literales numéricas***
  Se crean a partir de algún tipo de datos primitivo.
  Ejemplos
  ```
  747          // literal entera
  3.13.159F    // literal float
  ```

30030L // literal long

- ***Literales booleanas***
  Consisten de las palabras reservadas *trae y false.*

- ***Literales de caracter***
  Se manejan con un solo caracter y va entre apostrofes (comilla sencilla).

  Ejemplos
  'H' '$' '3'

- ***Literales de cadena***
  Son un conjunto de caracteres, en el caso de JAVA son objetos (instancias) de la clases ***String***, teniendo métodos parecidos a los de C/C++, para combinar, probar y modificar cadenas con facilidad y se representan por una secuencia de caracteres, las cuales van entre comillas (dobles)

  Ejemplo
  "Hola Contaduría" "Feliz Año Nuevo 2006"

Expresiones y operadores
Las ***expresiones*** son combinaciones de términos, o de variables, operadores y llamadas de métodos hechas de acuerdo a la sintaxis del lenguaje y que regresará un solo valor, del tipo que dependerá de los elementos usados.
Una ***expresiones*** es lo que se puede escribir a la derecha de una declaración de asignación.

Ejemplo

```
salida = "Hola Conrtaduría";

i = j = 3;                // Se pueden realizar asignaciones multiples
distancia = x2 – x1;      // Se pueden utilizar los operadores
```

Los ***operadores*** son los símbolos especiales que se encargan de realizar operaciones como; la suma, la resta, la multiplicación, etc.

La siguiente tabla 3.2 muestra un resumen de los distintos tipos de operadores.

Operador	**Significado o función**	**Ejemplo**
	Operadores aritméticos	
+	Suma	a + b
-	Resta	a - b

*	Multiplicación	a * b
/	División	a / b
%	Módulo (es el residuo de dos enteros)	a % b

Operadores de asignación

=	Asignación (es de derecha a izquierda)	a = b
+=	Suma y asignación	a += b (a=a + b)
-=	Resta y asignación	a -= b (a=a - b)
*=	Multiplicación y asignación	a *= b (a=a * b)
/=	División y asignación	a / b (a=a / b)
%=	Módulo y asignación	a % b (a=a % b)

Operadores de relación

==	Igualdad	a = = b
!=	Distinto	a != b
<	Menor que	a < b
>	Mayor que	a > b
<=	Menor o igual que	a <= b
>=	Mayor o igual que	a >= b

Operadores especiales

++	Incremento	a++ (postincremento) ++a (preincremento)
- -	Decremento Postdecremento.- primero evalúa y luego decrementa Predecremento.- primero decrementa y luego evalúa	a-- (postdecremento) --a (predecremento)
(tipo)expr	Cast (permite hacer la conversión entre tipo de datos primitivos)	a = (int) b
+	Concatenación de cadenas (permite unir cadenas)	a = "cad1" + "cad2"

.	Acceso a variables y métodos (es el operador ***punto)***	a = obj.var1
( )	Agrupación de expresiones, para cambiar las prioridades de evaluación	a = (a + b) * c
,	Operador coma **,**	int a,b,c;

TABLA 3.2 ***Operadores en JAVA***

La siguiente tabla 3.3 muestra el orden o la precedencia que se les dio a los operadores, los cuales aparecen en ese orden (de precedencia), los operadores que vienen en el mismo renglón tienen igual precedencia (prioridad).

Operador	**Notas**
[ ] ( )	Los corchetes [ ] son para los arreglos
++ -- ! ~	! es el NOT lógico (de negación) y ~ es el operador de bits de complemento **a r**
new (tipo)expr	new se utiliza para crear instancias de clases
* / %	Multiplicación, División y Módulo
+ -	Aditivos y de Substracción
<< >> >>>	Corrimiento de bits a la derecha y a la izquierda
< > <= >=	Relacionales
= = !=	Igualdad y diferente
&	AND (operador de bits)
^	OR exclusivo (operador de bits)
|	OR inclusivo (operador de bits)
&&	AND lógico
||	OR lógico
? :	Condicional Ternario
= += -= *= /= %= &= ^= |= <<= >>= >>>=	Asignación

TABLA 3.3 Precedencia de los o*peradores en JAVA*

Nota: Las expresiones, son evaluadas de izquierda a derecha y ya una vez evaluadas, ese valor se pasa a la variable (identificadores) que se encuentra a la izquierda de la expresión. Es de hacer notar que los operadores de bits (como en C/C++), tienen la misma prioridad.

Variables

Las ***variables*** son localidades de memoria, en las que se almacenan los datos, cada una tiene su propio *nombre, tipo y valor*. Los tipos de variables que maneja JAVA son de:

- *Instancia.*-Se ocupan para definir los atributos de un objeto.
- *Clase*.- Son variables en las que sus valores son idénticos para todas las instancias de la clase
- *Locales.*- Son las variables que se declaran y se utilizan cuando se definen los métodos.

El *nombre* de la variable será algún identificador válido (de acuerdo a lo explicado anteriormente) y con el se hace la referencia a los datos que contienen la variable.

El *tipo* de una variable indica que valores puede manejar y alas operaciones que se pueden hacer con ella, por ejemplo si es una variable de tipo entero, se pueden realizar con ella las operaciones aritméticas, en tanto que si fueran de tipo carácter, no se podrían hacer este tipo de operaciones.

El *valor* que puede tomar es en sí la *literal,* que va a depender precisamente de su tipo.

Para indicar el tipo y un nombre de una variable, se sigue la siguiente sintaxis:

```
TipoDeLaVariable    nombreDeLaVariable;
```

Ejemplo

```
String letrero;          // variable de tipo string
char letra;              // variable de tipo char
int entero;              // variable de tipo entero
float real;              // variable de tipo float (variable real)
boolean logico;          // variable de tipo lógico (verdadero o falso)
Avion boing;             // se declara una variable llamada boing que es de tipo
                         // Avion

Vuelo comercial, militar, privado;   // se declaran varias variables del mismo
                                     // tipo, se declaran las variables comercial,
                                     // militar y privado los tres de tipo Vuelo

String  mensaje = "Hola Contaduría"; // se declara la variable mensaje de tipo
                                     // string y se inicializa al momento de su
```

// declaracion

Las variables como se mencionó anteriormente pueden ser de los siguientes tipos:

Primitivo (número, carácter o booleano).

En la siguiente tabla 3.4 se muestra un condensado de este tipo de variables.

Tipo	Descripción	Tamaño/Formato
	Números enteros	
byte	Entero byte	8-bit
short	Entero corto	16-bit
int	Entero	32-bit
long	Entero largo	64- bit
	Números reales	
float	Punto flotante	32-bit
double	Punto flotante de doble presición	64-bit
	Otros tipos	
char	Un solo carácter	16-bit caracteres Unicode
boolean	Un valor booleano	true o false

TABLA 3.4 Datos Primitivos *en JAVA*

Referencia
En JAVA ya no existen los apuntadores, pero se sustituyen con las *referencias,* el valor de una variable de tipo referencia tiene una dirección de un grupo de valores representados por una variable.

Ejemplo

```
Atleta nadador;     // se hace la referencia a un objeto  de la clase Atleta

String letrero;     // se hace la referencia a un objeto de la clase string
```

III.2.- Estructuras de control
En este punto se va a profundizar en lo que son las sentencias, vistas con anterioridad en el punto **III.1.**

Sentencia.- También llamada instrucción, es la unidad básica del lenguaje, es la tarea más sencilla que realiza un programa. Existen diferentes tipos de sentencias como: *sentencias de expresión, de declaración de variables y de flujo,* que son las que se detallaran más adelante.

- ***Sentencias de expresión***
  Recordando que las expresiones terminan con punto y coma ; y a continuación, se presentan algunos ejemplos de tipos diferentes de sentencias.

  Ejemplos:

```
pi = 3.14159;                          //  sentencia de asignación
a++;                                   //  sentencia de posincremento
--b;                                   //  sentencia de preincremento
System.out.println("Hola Contaduría");       // sentencia de llamada a un método
Avion jet = new Avion( );                    // sentencia para la creación de objetos
```

- ***Sentencias de declaración de variables***
  Son para declarar variables.

  Ejemplos

```
int  b;
double  a = 8933.234;          // se puede inicializar el valor de la variable
String mensaje;
a=b=5;          // se pueden inicializar mas de una variable al momento de
                // declarar las variables
```

- ***Sentencias de control de flujo***
  Estas sentencias determinan el orden en el cual serán ejecutadas otro conjunto de sentencias, entre las más comunes están las siguientes:

  - Las sentencias *condicionales* como: el ***if then else, el switch.***
  - Las de *iteración o bucles como:* ***el for, el while, el do while.***

  Las sentencias de control de flujo y en general las sentencias son muy parecidas a las que se ocupan en C/C++, a excepción de que JAVA no tiene el ***goto.***

A continuación se presentan más detalladamente estas ***sentencias de control de flujos.***

La sentencia condicionales

if then else
La sentencia **if** evalúa en primer lugar su expresión condicional, la cual solo podrá tener dos valores: *verdadero o falso* y dependiendo de ello lleva la ejecución de un grupo de sentencias o de otro grupo de ellas o ninguna sentencia. Con esta sentencia se lograra que la computadora tome decisiones, basándose en el criterio de la condición dada.

La sintaxis en general del if then es:

```
if (condición)
{
        Conjunto  de sentencias;
 }
```

En este caso solo se trata del if then donde únicamente se realiza(n) la(s) sentencia(s), cuando la condición que se evalúa es verdadera, y en caso contrario cuando es falsa no se realiza nada y continua con el flujo del programa después del if. Es de hacer notar que lleva las dos llaves de { y } para delimitar el bloque de instrucciones que se ejecutaran cuando sea verdadera la condición, pudiendo ser opcionales, cuando se trate de una sola instrucción.

La sintaxis del if then else es:

```
  if (condición)
{
        Conjunto  de sentencias grupoa;
}
else
{
        Conjunto  de sentencias grupob;
}
```

La diferencia con el if then es que aquí si se ejecuta el conjunto de sentencias del *grupoa,* cuando la condición es verdadera, en tanto que si la condición es falsa se ejecutan las sentencias del *grupob.* En ambos casos se continúa la ejecución del programa después del if.

La sintaxis del if anidado es:

```
if (condicion1)
  {
  instruccion;
  }
else
  if (condicion2)
    {
    instruccion;
    }
    .
    .
    .
    .
  else
      {
      instruccion;
      }
```

Se pueden emplear, tanto if then, como if then else.

La sintaxis del if else if es:

```
if (condicion1)
  {
  instruccion;
  }
else if (condicion2)
    {
    instruccion;
    }
    .
    .
    .
    .
  else
       {
       instruccion;
       }
```

Y también se pueden emplear, tanto if then, como if then else. Es de hacer notar, que para cualquiera de las sintaxis anteriores del if, no es necesario dejar las sangrías, que se indican en los renglones anteriores, pero se recomienda dejarlos, para mayor claridad en la programación.

Otra variante del if es la alternativa ? (condicional ternario) y su sintaxis es:

exp1 ? exp2: exp3

Donde exp1 se evalúa y si resulta verdadera, se evalúa exp2 y si es falsa exp1, entonces se evalúa exp3.

Ejemplos.

```
if (valor > 0)                    // ejemplo del if then con una sentencia
        System.out.println("Hola Contaduría");
```

```
if (valor > 9)          //ejemplo del if then else con varias sentencias
{
        System.out.println("Hola Contaduría");
        System.out.println("Hola Administración");
}
else
```

```
{
        a=3;
        System.out.println("a=3");
}


a=20;                   // ejemplo del condicional ternario
b=a= = 20?10:20;
System.out.println ("b quedo con el valor de 10");
```

Nota: el System.out.println es una de las formas más sencillas para desplegar resultados.

La sentencia switch

Es una estructura de selección y se utiliza cuando se quiere comparar una variable (solo de tipo entero o char) con una serie de valores diferentes, se indican los posibles valores que puede tomar la variable y las sentencias que se ejecutarán cuando la variable coincide con alguno de estos valores.

Para el switch se tiene la siguiente sintaxis:

```
switch (variable o expresión)
{
case cte1:
        bloque de sentencias
        break;
case cte2:
        bloque de sentencias
        break;
case cte3:
        bloque de sentencias
        break;
  .
  .
  .
.
case cten:
        bloque de sentencias
        break;
default:
        bloque de sentencias
}
```

Su función es la de realizar solo el grupo de sentencias correspondientes de acuerdo al valor de la variable que deberá de evaluarse. Es como si se tratara de varios ifs anidados. A grandes rasgos su funcionalidad se puede resumir en los siguientes puntos:

- Solo puede evaluar la expresión por igualdad.

- Solo puede evaluar expresiones enteras y si se utilizan de tipo carácter se convierten automaticamente a sus valores enteros.
- El uso del ***break***, indica que salga de la instrucción ***switch*** transfiere el control al final de la sentencia ***switch***, es por ello que el default, puede o no llevar break, ya que es la última parte del switch. Si se desea que más de un case se realice, se pueden omitir los breaks, ya que se saldrá de este switch hasta que encuentre un ***break o*** termine todo el cuerpo del sitwch.
- Puede tener otros sitwch anidados en alguno de los cases, en donde si se repita el valor de la constante.
- Si se desea que para más de una opción de case, se ejecute lo mismo, bastara con no ponerle el ***break***.
- El ***default*** (su bloque de sentencias), lo realizará solo cuando no se cumpla ninguna de las condiciones que se indicaron con cada uno de los cases.

Ejemplo:

```
int a=4;
switch(a)
{
   case 3:
            printf("I");
   case 2:
            printf("I");
   case 1:
          printf("I");
            break;
   case 4:
            printf("I");
            printf("V");
          break;
   default:
            printf("no es un numero que este entre 1 y 4");
 }
```

Las sentencias de iteración o bucles

Permiten ejecutar un número finito (si es que no se llega a un “loop”) de veces un grupo de sentencias, hasta que se llega a cumplir cierta condición.

El ciclo for

Logra repetir una sentencia o un bloque de sentencias mientras la condición se cumpla. Para el ***for***, su sintaxis es la siguiente:

```
for (variable de inicio; condición ; acción de incremento o decremento)
{
        grupo de sentencias
}
```

La inicialización que se da en la *variable de inicio,* es una sentencia de asignación que se utiliza para establecer una variable que controle el ciclo.

La condición es una expresión que prueba la *variable de inicio* que controla el ciclo y determinar cuando salir de él.

El *incremento o decremento* es el que hace que cambie la *variable de inicio,* para que en algún momento dado pueda salir del ciclo.

Los ciclos while y do while

Estos ciclos son muy parecidos al ***for*** solo que tienen algunas ligeras variantes ya que repite un bloque de sentencias mientras se cumpla con la condición, sobre todo en su sintaxis, a continuación se describen más a detalle.

Para el do while su sintaxis es:

```
do
{
        conjunto de sentencias

} while (condición);
```

Es un ciclo repetitivo, que se va a realizar lo que se encuentra dentro de los delimitadores { y } , mientras se cumpla la condición. Primero ejecuta el conjunto de sentencias y luego evalúa la condición**.**

Para el while se tiene la siguiente sintaxis:

```
while (condición)
{
        Conjunto de sentencias
}
```

Su función es como el do while, solo que aquí, primero pregunta (evalúa) y luego

ejecuta.

break y continue

El ***break*** es una de las instrucciones de salto, que cuando se ejecuta, se va a la siguiente instrucción, esto es; fuerza la terminación de la instrucción, de donde apareció el ***break***, sin hacer caso de las líneas que están después del ***break.***

Se utiliza en el ***switch*** dentro de los ***cases*** y también se utiliza para la terminación inmediata de un ciclo, brincando la condición de terminación del ciclo.

El ***continue***, también es una instrucción de salto, pero continúa la acción de la instrucción, ya que fuerza una nueva iteración del ciclo, ignorando todo el código que aparezca a continuación**.**

return

Logra la salida del método actual de ejecución, regresando que el control del programa vuelva al código que lo llamó. Pudiendo devolver un valor, el cual va seguido después de la palabra reservada ***return.*** El tipo de este valor es del mismo que el que tiene la declaración del método a excepción del método que sea declarado ***void*** y entonces no regresará ningún valor, como sucede en C/C++.

Ejemplo:

```
return raiz;
```

Constructores, creación de objetos y recolección de basura

Una vez que se ha definido la *clase,* se pueden crear *objetos,* de esa *clase,* para lo cual es necesario declararlos, de forma similar a como se declaran variables de tipo de datos sencillos, pero ahora se usa la clase como tipo de objeto, es similar a la sentencia ***struct y malloc*** de C.

Creación de objetos.

En la creación de objetos se utiliza la palabra clave ***new***, es de hacer notar que no siempre se deben de declarar los objetos antes de utilizarlos, ya que existen clases predeterminadas como ***String.***

Ejemplo:

En la línea siguiente se realiza al mismo tiempo la creación y la referencia del objeto

```
Vuelo avion=new Vuelo();
```

En las siguientes dos líneas ser realizan por separado la creación y la referencia del objeto.

```
Vuelo avion;          // Se crea la variable objeto (memoria stack)
                      // es en sí la declaración del objeto
avion = new Vuelo(); // Se realiza la referencia del objeto
                      //(memoria heap). Se inicializa el objeto
```

El operador new

new crea una instancia de clase proporcionando la memoria necesaria al objeto creado, este operador esta vinculado con el *constructor*.

Constructores.

Como se comentó en el capítulo anterior, es un tipo específico de método, cuyo nombre es el mismo nombre que la clase. Y que es utilizado cuando se quieren crear objetos de esa clase y es utilizado o ejecutado al crear e iniciar dicho objeto.

Puede existir más de un constructor y se conocen como *constructores múltiples,* con o sin parámetros. Si no existe algún constructor, JAVA proporciona uno por omisión, el cual inicializa las variables del objeto con los valores que se dan predeterminadamente.

Recolección de basura

Es un proceso que tiene JAVA, para reclamar el espacio de memoria de un objeto, cuando ya no exista ninguna referencia al objeto.

Manejo de métodos y variables

Recordando que un objeto puede tener *atributo o características y métodos o acciones*, va a ser necesario acceder a ellos a través del operador **punto(.),** de las siguientes maneras, para poder trabajar con estos elementos del objeto.

- *Manejando sus variables* (atributos) en forma directa. Cuya sintaxis es:

```
nombreDelObjeto.nombreDeLaVariable;
```

- *Utilización de sus métodos* (acciones).- Su formato es:

nombreDel Objeto.nombreDelMétodo (<lista de argumentos>);

NOTA: recordando que lo que se encuentra entre pico paréntesis es opcional.

Ejemplo:

```
/* Programa creado con el nombre de Alumno.java */

class Alumno
{
    int numCta;             // Se declaran las variables (atributos)
    String nombre;           // de la clase Alumno
    int edad;
    String direccion;
    int cveCarrera;

    Alumno()              //Primer constructor de la clase Alumno
    {                          //(se pueden tener varios constructores)
        numCta=0;          //cuya función en este ejemplo es la de
      nombre = null;       //inicializar las variables de la clase
      edad = 0;            //no tiene ningún parámetro
      direccion = null;
        cveCarrera = 0;
    }
```

```
Alumno(int numCta, String nombre, int edad, String direccion,
     int cveCarrera)  // se delcara el segundo contructor
{
  this.numCta = numCta;   //   la palabra clave this que se
  this.nombre = nombre;   //   refiere solo a las varibales
  this.edad = edad;       //   de este método
  this.direccion = direccion;
  this.cveCarrera = cveCarrera;
}

void despliegaDatos (int numCta, String nombre) /* Se utiliza el Polimorfismo (por
              sobrecarga) con este método que tiene dos formas, este es con
                                  parametros */
{
  System.out.println("\nEn el metodo con parametros los datos son:");
  System.out.println("Cta = " + numCta  + "   Nombre:"   +
        nombre);
}


void despliegaDatos ()     /* Se utiliza el mismo método ahora
                  sin parámetros
                     Sytem.out.println permite desplegar mensajes y
```

```
                    datos en el monitor   y acepta los caracteres de escape
                    pra el salto de líneas, que se utilizan en C*/
  {
      System.out.println("\n\nEn el metodo sin parametros los datos son:");
      System.out.println("Cta = " + numCta  + "  Nombre: "   +
          nombre  + "  Edad: " + edad + "  Direccion: " +
          direccion + "  Cve de la carrera " +
          cveCarrera);

  }

 public static void main(String args[]) /* Se declara el método
                                          principal */
 {
   Alumno alUno = new Alumno();/* Se utiliza el primer constructor
*/

   Alumno alDos = new Alumno (745, "Luis Arenas", 24, "Conocida",22);
                    /* Se utiliza el segundo constructor con
                                    Parámetros  */
   alUno.numCta=999;    /* Se inicializan algunas de las variables*/
   alUno.nombre="Juan Perez";
  // alUno.direccion="Palaza de la Constitucion 130";
   alUno.despliegaDatos(alUno.numCta, alUno.nombre);    /* se llama
          al método despliegaDatos con parámetro el objeto)*/
```

```
    alDos.despliegaDatos();  /* se utiliz al método depliegaDatos
                                a traves del objeto alDos   */
  }
}
```

La salida del programa es:

En el método con parámetros los datos son:

Cta = 999 Nombre: Juan Perez

En el método sin parámetros los datos son:

Cta = 745 Nombre: Luis Arenas Edad: 24 Dirección: Conocida Cve de la carrera 22

Herencia de clases en JAVA

Retomando el concepto de ***herencia*** visto en el capitulo anterior, se mostrará como se utiliza en JAVA, recordando que este concepto esta muy vinculado a la estructura jerárquica del árbol, donde se organizan las relaciones entre las clases.

En JAVA se maneja solo la ***herencia simple,*** por lo que cada clase solo tiene una clase padre, la cual se conoce como ***superclase*** y a la clase hija como ***subclase***. Una ***subclase*** hereda las variables y métodos de su superclase, aunque puede llegarse a dar el caso de que se anulen o se agreguen algunos de estos elementos.

Para implementar una subclase, se utiliza la palabra clave **extends** en la declaración de la clase. Su sintaxis es la siguiente:

```
class nombreDeSubclase extends nombreDeSuperclase
{

}
```

En JAVA se puede implementar la herencia múltiple (cuando una subclase tiene más de una clase padre), de dos formas:

- Declarando varias herencias simples.
  Por ejemplo. La clase ***a*** sea subclase de ***b*** y de ***c,*** se puede declarar las siguientes herencias simples.

  ```
  class B
  ```

```
{
        // cuerpo de la clase B
}

class C
{
        // cuerpo de la clase C
}

class A extends c
{
        // cuerpo de la clase A
}
```

- A través de *Interfaces.*- Las cuales se detallaran más adelante.

Es de hacer notar, que la madre de todas las clases en JAVA es la Object (superclase de todas las clases).

Sobre escritura de métodos

Una subclase hereda todos los métodos de su superclase a excepción de que la subclase sobrescriba los métodos.

Se dice que una subclase sobrescribe un método de su superclase cuando crea un método con las mismas características de nombre, número y tipo de argumentos, que el método de la superclase. Y se emplea generalmente para agregar, quitar o modificar la funcionalidad del método que se hereda de la superclase.

Ejemplo:

```
class A
{
  void  metodoUno(String var)
  {
        // Cuerpo del metodoUno
  }

  void metodoDos( )
  {
        //  Cuerpo  del metodoDos
  }
}

class B extends A
{
         /* Estos métodos sobrescriben a los métodos  de la clase padre */
           void metodoUno (String var)
        {
```

```
            // Cuerpo del metodoUno modificado en la subclase B
        }

        void metodoDos( )
         {
            // Cuerpo del metodoDos modificado en la subclase B
        }
}
```

Tipos de Clases (abstractas, común y final)

Clase abstracta.- Recordando el árbol jerárquico de las clases, la raíz de todas ellas se conoce como clase *abstracta* y es la clase que declara la existencia de métodos pero no la implementación de dichos métodos, esto es no lleva las llaves { } y las sentencias entre ellas.

En una clase abstracta por lo menos uno de los métodos debe ser declarado abstracto.

Se utiliza la palabra clave **abstract**, para declarar una clase o un método como abstractos.

Ejemplo:

```
abstract class A
{
  abstract void metodoUno(String  var1);
  void metodoDos( )
        {
            // Cuerpo del metodoDos
        }
}
```

De una clase abstracta no se puede crear objetos, pero si se puede heredar y las subclases, podrán agregar la funcionalidad a los métodos abstractos. Puede darse el caso de que si no lo hacen así, las subclases serán también abstractas.

Una clase *común,* es la parte del árbol jerárquico que se encuentra entre en los niveles intermedios de dicho árbol. En esta clase se puede heredar y crear objetos. Se tiene ascendencia y descendencia.

En la clase *final,* no se puede heredar, se tiene ascendencia, pero no descendencia y es el nivel que se encuentra mas abajo del árbol jerárquico. Y se utiliza la palabra clave ***final.***

Ejemplo:

```
        final  class animal
        {
            //cuerpo de la clase animal
        }
```

Interfaces

Una ***interface*** es un conjunto de constantes y de métodos abstractos La definición de una ***interface*** es prácticamente igual que el de una clase, ya que solo se sustituye la palabra clave ***class*** por . ***interface.***

Ejemplo:

```
interface Mamifero
{
        // Cuerpo de la interface Mamifero
}
```

Con la plabara clave ***implements,*** se puede hacer uso de una o de varias ***interfaces***, con lo cual se puede simular la ***herencia múltiple,*** que se mencionó anteriormente,

Ejemplo:

```
class Gato implements Mamifero
{
        // Cuerpo de la clase Gato
}
```

Ejemplo:

```
class Rana extends Invertebrados implements Pescados, Anfibios.
{
        // Cuerpo de la clase Rana que heredo de tres clase diferentes (Invertebrados
        // Pescados y Anfibios
}
```

NOTA: Se pueden anular o sobrescribiendo nuevamente los métodos, en las subclases.

III.3.- Paquetes de JAVA

Un paquete es un grupo de ***clases e interfaces*** relacionadas y sirve para lograr que la clase sea más fácil de localizar, logrando también un mejor control de acceso a los miembros de una clase. Se pueden utilizar paquetes ya desarrollados por JAVA o bien por los realizados por el mismo programador o por terceros.

Creación de paquetes

Si se desea realizar que alguna ***clase o interface*** pertenezca a un paquete se utiliza la siguiente sintaxis:

package *nombre*;

donde *nombre* puede ser tener una sola palabra o varias separadas por puntos, ya que esta compuesto como sigue dicho nombre:

nomDel Paquete.nombreDel SubpaqueteUno.nombreDelSubpaqueteDos…clase

Ejemplo 1:

```
package paquete1;
class ClaseUno
{
        //Cuerpo de la ClaseUno
}
```

Ejemplo 2:

```
package paquete2.apellido;   // Los nombres de los paquetes corresponden con el
class ClaseDos               // nombre de los directorios en el sistema de archivos
{
        // Cuerpo de la ClaseDos
}
```

Si se desea utilizar una clase en especial, se utiliza la palabra reservada ***import.***.

Ejemplo:

```
        import java.util.Date;
```

Si se ahora se requiere utilizar todas loas clases de un paquete especifico, se utiliza el asterisco.

Ejemplo:

```
        Import java.util.*;
```

El orden en que van apareciendo en un archivo fuente de una aplicación de JAVA estos elementos que se están tratando es el siguiente:

```
package contaduría.informatica.software;      //Se le da el nombre al paquete que se va a
crear

import java.util.Date;                        //Se declaran los paquetes que se van a
utilizar
import java.awt.Panel;                                //en esta aplicación

class Lenguajes4GL extends Panel              //Se utiliza solo el nombre de la clase
importada
                                              //ya que se importo solamente esa en
particular
                                              //del paquete más grande java y dentro de el
esta
                                              //el paquete awt.
```

```
                                        //Si se hubiera importado un conjunto de
clases
                                        //seria necesario especificar todo el nombre
del
                                       //paquete, subpaquete(s) y clase
{
        // Cuerpo de la clase Lenguajes4GL
}
```

Otro programa puede utilizar este paquete haciendo la referencia a el de la siguiente manera:

```
        import contaduría.informatica.software.Lenguajes4GL;
```

Un punto importante, es que la importación de paquetes y de clases, indica únicamente al compilador de JAVA donde buscar el código que necesita, sin aumentar el tamaño del programa.

También se puede importar una clase hecha por el mismo programador o por otra persona como la clase Salida que se define a continuación:

```
class Salida
{
        // Cuerpo de la clase Salida
}
```

En un programa se puede importar la clase Salida, como se muestra enseguida:

```
import Salida;
class EjemploUno
{
        // Cuerpo de la clase EjemploUno donde se ocupa algún(nos) métodos de la
        //clase que  se importó
}
```

Es de hacer notar que la clase que se importa debe de estar en el directorio actual de trabajo, si no va a mandar un mensaje de error, pero también se pueden organizar los programas de acuerdo por ejemplo, a su aplicación en diferentes directorios y a través de la importación se pueden compartir y utilizar en diferentes programas, aunque no se encuentren en el directorio actual de trabajo, pero se deberá de utilizar la variable de entorno de **CLASSPATH,** para indicarle al compilador de JAVA, en que otros directorios busque los paquetes, su sintaxis es:

```
        SET CLASSPATH =C:\directorio1;C:\directorio2;%CLASSPATH%
```

Se pueden añadir varios directorios donde busque el paquete, únicamente indicándolos en esta instrucción separados por **punto y coma (;)** y no dejando espacios en blanco junto al signo igual.

La versión 2 de JAVA, presenta un conjunto de cerca de sesenta paquetes, los cuales están dentro de del paquete general llamado ***java.*,*** entre los más importantes se encuentran:

- ***java.lang***.-Tiene las clase con las que puede trabajar el programa principal.
- ***java.util***.-Se encuentran clases especializadas como son la de los calendarios.
- ***java.io***.-Proporciona un archivo independiente del dispositivo y servicios de Entrada/Salida.
- ***java.awt***.-Contiene la mayoría de las clases dedicadas a los gráficos. Este paquete es de vital importancia y se detallará más a detalle en el siguiente capítulo.
- ***java.net***.-Son las clases que pueden trabajar con los programas de bajo nivel en Internet y WWW.
- ***java.applet***.-Cuenta con una clase que puede trabajar con el lenguaje HTML, el cual se utiliza en los applets de JAVA. Este paquete también se describirá más a detalle en el capítulo cinco.

III.4.- Programación con excepciones

Un punto trascendental en la elaboración y ejecución de programas es el manejo de errores, los cuales pueden ser detectados en dos fases:

- En la compilación, que serían los errores de sintaxis y de estos brinda mucha ayuda el compilador, ya que indica el número de la línea donde se detecto el error, así como el mensaje de error.
- En la ejecución, donde la mayoría de esos errores serían los de semántica o lógicos, como por ejemplo la división entre cero, que el compilador en muchos de los casos no lo detecta, ya que puede ser algún datos que lea el programa el que se utilice, para realizar esa división y que en algún momento pueda llegar a ser cero. Si no se detectan a tiempo estos errores, puede hacer que el programa termine abortando su ejecución, dando resultados que no son los deseados, si es que llega a darlos, y en algunos casos, se puede llegar a perder inclusive información.

Por lo que es conveniente, que varios de estos errores sean manejados en el momento de la ejecución del programa, para que esta no se vea afectada.

Es necesario asilar la posible causa de error y manejarla, de tal forma que no cause problemas, ya sea indicando haga “algo” y que el programa aborte su ejecución o que prosiga su ejecución.

Haciendo una analogía, casi es como el manejo de “interrupciones” cuando se programa en ensamblador.

Cuando sucede una excepción impide la continuación de la ejecución de una parte del programa y no puede continuar, porque no se tiene la información suficiente para solucionar ese "problema o situación inesperada".

JAVA permite seleccionar la parte del código donde se contemple que pueden suceder situaciones "inesperadas", así como cuales podrían ser y en cada una de ellas se indica que acciones se van a seguir.

Las excepciones en Java son objetos y tienen su propio árbol de jerarquía. La clase raíz de ellas es Throwable, que es una subclase de Object (la clase madre en JAVA). Los métodos que se definen para estos objetos serán los que manden los mensajes error que estén relacionados con cada uno de los diferentes tipos de excepciones. En la figura III.1 se muestra este árbol a grandes rasgos:

FIGURA III.1 ***Árbol de excepciones en JAVA***

De esta figura se puede apreciar lo siguiente; los errores y las excepciones son subclases de ***Throwable,*** por lo que heredan sus métodos, entre los que se encuentran:

- **toString( )** muestra el nombre de una excepción junto con el mensaje que devuelve getMessage().
- **getMessage( )** se utiliza para obtener un mensaje de error asociado con una excepción.
- **printStackTrace ( )** permite imprimir el registro del snack donde se inició la excepción.

III.5.- Cláusula try match

Cuando se activa una excepción, se "lanza", alguien debe de capturarla y hacer "algo". Se utilizan los controladores de excepción para determinar en un momento la excepción que se ha "lanzado"El manejo de excepciones en JAVA permite separar el código del problema a resolver, del código de errores que se puede generar.

Se delimita un bloque a través de la palabra clave ***try*** dentro del método, para aislar el código que puede generar una excepción, esto es para especificar el bloque de declaraciones cuyas excepciones serán controladas por medio de una serie de cláusulas ***catch***, las cuales pude variar en cuanto a su número. Esto quiere decir que un mismo bloque delimitado por ***try*** puede tener varios ***catch's***, que detecten diferentes causas de excepción y diferentes acciones a seguir.

Cuando se lanza o tienen lugar una de estas excepciones, entonces se ejecutará el bloque de la excepción, que coincida con el de la clase o superclase de la excepción.

Su sintaxis es:

```
try
{
        //cuerpo del bloque que puede generar excepciones
}
```

Cuando una excepción es lanzada, se debe de manejar por medio de otro bloque de código aparte, conocido como *el manejador de excepciones*, existiendo un bloque por cada excepción diferente que se maneje. Estos manejadores de excepciones se distinguen por llevar la palabra clave ***catch.***

Cada bloque de código de ***catch*** funciona como un método de un solo parámetro o argumento, siendo éste el que indica que tipo de error o excepción es con el que se activa. La sintaxis general es:

```
try
{
        //cuerpo del bloque que puede generar excepciones
}
catch(nombreDel Error1 identificador1)
{
        // Método que maneja la excepción del nombreDel Error1
}
catch(nombreDel Error2 identificador2)
{
        // Método que maneja la excepción del nombreDel Error2
}
.
.       //Tantos catch's como sean necesarios
.       //(número de excepciones que se puedan generar)
```

También se puede atrapar cualquier tipo de excepción, a través de la clase base ***Exception*** su sintaxis es la siguiente:

```
Match (Exception e)
{
        // Bloque que maneja la excepción en general
}
```

Dentro del bloque se pueden manejar algunos de los métodos de la superclase de ***Exception,*** que es ***Throwable,*** como se puede apreciar en la figura III.1.

Declaración throwable,
Se puede lanzar una excepción a través de la declaración ***throw*** su formato se muestra a continuación:

throwable expresión;

Siendo expresión la que se valore con la clase ***Throwable*** o en su subclases y en la mayoría de los casos se utiliza con ***new.***

Ejemplo:

throw new IOException("Mensaje de error");

IV. PROGRAMACIÓN DE INTERFASES GRÁFICAS

A principios de la década de los noventas, es cuando Internet se empieza a popularizar, al pasar al ámbito universitario y comercial, en esa época, se utilizaban en los navegadores más populares de esa época como Mosaic, solo presentaciones de texto estático, posteriormente imágenes sin movimiento y a últimos años, elementos de sonido y animaciones, cada vez se usan más los ambientes gráficos.

Es precisamente por principios de la década pasada, que es cuando nace JAVA, logrando satisfacer en gran medida los requerimientos que Internet va exigiendo, con todos estos elementos, que hacen que de paginas estáticas, se logren paginas más atractivas, con movimiento e interactivas.

En la versión de JAVA 1.0 se diseñó la interfaz gráfica de usuario (GUI),la cual da la facilidad de construir aplicaciones gráficas, compatible con "cualquier" plataforma. Posteriormente se logró mejorar estas aplicaciones con la creación del ***awt (A***bstaract ***W***indow ***T***oolkit) yes en la versión de JAVA 1.1 en que se perfecciono el ***awt*** y además se le incorporó otra herramienta que se conoce como Java Beans, la cual tiene componentes para la creación de ambientes visuales.
Se consolida este ambiente gráfico en la versión de JAVA 2, que es con la que se prepararon los ejercicios, que se muestran en este trabajo.

IV.1.- El paquete JAVA.awt

Como se mencionó en el capítulo anterior ***java.awt,*** contiene la mayoría de las clases dedicadas a los gráficos y es el más grande e importante de todos los de JAVA. Se dividen los siguientes grandes grupos:

- El dedicado a los componentes GUI (interfaz gráfica de usuario) y son:
  - El subárbol ***Component.***
  - El subárbol ***MenuComponent.***
- El de clases especializadas en la gestión de capas.

- ***FlowLayout.***
- ***BordeLayout.***
- ***CardLayout.***
- ***GrildLayout.***
- ***GridBagLayout y GridBagConstraints.***
- ***Instes.***

- El de clases gráficas.
  - ***Graphics, Graphics2D y PrintGraphics.***
  - ***Image.***
  - ***Color y SystemColor.***
  - ***Font.***
  - ***FontMetrics.***
  - ***AlphaComposite y Composite.***
  - ***BasicStroke y Stroke.***
  - ***GraphicsConfigTemplate, GraphicsConfiguration, GraphicsDevice y GraphicsEnvironment.***
  - ***GradientPaint y TexturePaint.***
  - ***RenderingHints,***
- El de clases geométricas.
  - ***Point.***
  - ***Polygon.***
  - ***Dimension.***
  - ***Rectangle.***
  - ***Shape.***
- El de clase por eventos
  - ***Event.***
  - ***AWTEvent.***
  - ***AWEventMulticaster.***
  - ***EventQueue.***
  - ***ActiveEvent.***
- Y el que tiene todas las demás clases varias.
  - ***MediaTracker.***
  - ***Toolkit.***
  - ***PrintJob.***
  - ***Cursor.***
  - ***ComponentOrientation.***

IV.2.- Componentes y elementos gráficos

Cuando se desean realizar animaciones, se utilizarán diferentes clases como la GUI, entre las características que se manejarán son las de los colores e imágenes.

En el párrafo anterior únicamente se mencionaron las clases graficas de ***awk***, en este punto se describirán con un poco más de detalle las más importantes de ellas.

Graphics, Graphics2D y PrintGraphics

La clase ***Graphics*** es de tipo abstracta y puede manejar modelos en ***2D***, sus métodos más importantes pueden realizar funciones como:

- Dibujar Texto.

- Copiar áreas rectangulares.
- Dibujar figuras o superficies tales como; rectángulos, óvalos, polígonos y arcos.
- Trazar líneas.
- Traslación del sistema de coordenadas.
- Recortar rectángulos.
- Cambiar el color.
- Funciones para solicitar gráficos.

La clase ***Graphics2D*** es una extensión de ***Graphics*** con la particularidad, que puede desarrollar un modelo en ***2D*** más complejo, ya que entre lo que puede realizar esta lo siguiente:

- Dibujar las imágenes, al mismo tiempo que realiza una serie de transformaciones.
- Dibujar formas con características tales como el de presentar la vista previa, o con líneas de diferentes grosores, etc.
- Un manejo de texto, que permita el recortar una parte de él.
- Mover o trasladar el sistema de coordenadas.
- Funciones para solicitar gráficos.

La última clase de este grupo que es ***PrintGraphics,*** en realidad es una interfaz, que cuenta con todos los métodos de Graphics y direcciona el resultado de los comandos en la impresora, ya que lo más común es que se presenten en el monitor.

Image
Permite el manejo de imágenes independiente de cualquier plataforma, basándose en un mapa de bits (bitmaps), los métodos que tienen estas clases permiten realizar cosa como las que a continuación se mencionan:

- Pedir las propiedades de la imagen, tales como el formato, de los derechos del copyright, entre otros.
- Solicitar las dimensiones de una imagen.
- Y realizar un contexto gráfico para la imagen, que permita trabajar con los métodos de dibujo de ***Graphics***.

Color y SystemColor
Permite a través de una estructura de datos manejar de forma independiente a la plataforma, se pueden manejar varios métodos y entre las acciones que permite hacer se tienen las siguientes:

- Transformación entre los modelos RGB y HSB.
- Manejo de los componentes de los colores rojo, verde y azul.
- Modificar la intensidad del brillo de un color.

Font yFontMetrics
Permiten acceder y solicitarle al sistema de fuentes locales para la realización de funciones tales como:

- Determinar una familia de fuentes, estilo, así como del tamaño, dándolas por puntos.
- Pedir los atributos y medidas de una fuente, tales como el nombre, estilo, tamaño del punto, ancho , etc.

AlphaComposite y Composite
Composite es una interfaz, permite manipular las imágenes para darles efectos como el de la transparencia y ***AlphaComposite*** es el responsable de producir la transparencia.

BasicStroke y Stroke
La clase ***BasicStroke*** da la facilidad de trabajar con: el ancho del lápiz, los atributos de un boceto y de decoraciones utilizando líneas.
En tanto que ***Stroke*** es la clase que se encarga de describir el lápiz virtual, en la que el usuario dibuja en el monitor a mano alzada.

GraphicsConfigTemplate, GraphicsConfiguration, GraphicsDevice y GraphicsEnvironment
Estas clase permiten describer los destinos de las operaciones que son utilizada por ***Graphics2D***, ya que cada entorno gráfico esta compuesto de una serie de dispositivos, requiriendo cada uno de ellos su propia configuración.

GradientPaint y TexturePaint
GradientPaint, permite dar un gradiente lineal de color y ***TexturePaint*** proporciona una imagen, que será utilizada para rellenar el modelo.

RenderingHints
Esta clase ayuda a ***Graphics2D,*** para que trabaje con los atributos del dibujo original.

IV.3.- Modelo para el manejo de eventos
Un evento es "algo", que puede activar alguna reacción, para el caso del funcionamiento de una computadora, entre los que se tienen los siguientes.

- Del ratón, como lo son los diferentes clics del ratón.
- De movimientos del ratón, como el de arrastre.
- Clics de botones.
- Introducción de caracteres en un campo de texto.
- Y selecciones de opciones, con el empleo del ratón.

Para cada tipo de evento que se llegase a producir, el programa debe de contemplar un método especial que se denomina *manipulador de eventos*, para cada tipo de dicho evento.
Un *manipulador de eventos*, es una clase que implementa la interfaz apropiada al evento que aconteció.

Event.- Es la clase que permite delegar eventos.***AWTEvent.AWEventMulticaster.***

V. PROGRAMACIÓN DE APPLETS

Recordando del primer capítulo, los ***applets*** son programas pequeños, que son introducidos en una página Web y que requieren de un Navegador Web compatible con Java para poder ejecutarse. Es frecuente que los ***applets*** se descarguen junto con la página HTML desde un Servidor Web y se ejecutan en la máquina cliente.
Los ***applets*** son herramientas, que soportan la aplicación o arquitectura de Cliente/Servidor, que se puede utilizar en una red de computadoras.

Los ***applets*** estándar están construidos en la clase Applet, el cual se encuentra en el paquete java.applet, de donde derivan sus propias clases de ***applets*** con solo usar la palabra clave ***extends,*** como se muestra en el ejemplo que se muestra en los parrafos posteriores.

Ventajas y deventajas de un applet
Pero todo en la vida tiene sus ventajas y desventajas y lo un ***applet*** no es la excepción, a continuación se describen algunas de ellas:

Desventajas

- No puede tocar ningún disco duro o local.
- No puede ejecutar comandos del sistema operativo.
- Un ***applet*** debe de concertarse a través de un ***socket*** (el cual es uno de los extremos de comunicación entre procesos, en un entorno de red).

Ventajas

- Facilitan la construcción de aplicaciones en una red de computadoras (sobre todo en la arquitectura Cliente/Servidor).
- No se instala.- Funciona como una aplicación independiente de la plataforma.
- Actualización instantánea.- La actualización del ***applet*** es del lado del servidor y se ve reflejada cualquier actualización del lado del cliente.

V.1.- Declaración de un *applet*

De manera similar a como se crea una aplicación, se hace en el ***applet*** , a continuación se muestra un ejemplo:

- Se edita el código fuente.

```
import  java.applet.*;          //Se realiza la importacion de los paquetes
import  java.awt.*;             //del applet y de  los graficos

public class AppletHolaContaduria extends Applet
{
   public static void imprime( Graphics g)
   {
      g.drawString( “Hola Contaduria", 50, 50);  //se manda el
   }                                             // letrero y las
}                                                //coordenadas
```

- Se guarda en un archivo con extensión **java** y entre comillas, para que sea un archivo de **texto**

  Guardar como y en nombre del archivo se pondrá
  "AppletHolaContaduria.java"

- Se compilará el programa con el siguiente comando:

  javac AppletHolaContaduria.java

  Si se llegan a tener errores de compilación, en este momento los desplegará y si no se tiene ningún error, entonces se generará, un archivo con extensión **class.**

Se crea con algún editor de textos, de forma similar al archivo de extensión .java, uno en HTML, el cual incluirá el ***applet*** como el ejemplo que a continuación se muestra:

```
<HTML>
<HEAD>
        <TITLE>PRIMER APPLET</TITLE>
</HEAD>
<BODY>
        <BR>
        <CENTER>
        <APPLET
                CODE=AppletHolaContaduria.class
                WIDTH=200
                HEIGHT=150>
        </APPLET>
        </CENTER>
</BODY>
</HTML>
```

- Se guarda el archivo entre comillas y con extensión **.html** como se ilustra a continuación:

  Guardar como y en nombre del archivo se pondrá
  "AppletHolaContaduria.html"

- A través de un navegador que puede ser Netscape o Explored por ejemplo, se abrirá este ***applet*****,** dando la opción de FILE y se actualizará la pagina, ejecutándose y mostrará, para este ejemplo:

  Hola Contaduría

  Los archivos que se generaron fueron:

- ApplerHolaContaduria.java.

- HolaContaduria.class.
- AppletHolaContaduria.html.

V.2.- Ciclo de vida

Los ***applets*** utilizan "marcos de trabajo" conocidos como *framework* y su *superclase* es JApplet, pudiendo sobrescribir sus métodos. Existen varios métodos que controlan lo que podría considerarse el ciclo de vida de un ***applet*** como lo es su creación y ejecución en una página Web.

A continuación se describen brevemente los métodos ***init, start, stop, destroy, paint y update.***

init

Es el primer método a llamar, es donde se inicializa el ***appet*** y sólo se llama una vez

start

Es llamado después de ***init,*** dicho método es llamado cada vez que un ***applet*** aparece nuevamente en la pantalla, por ejemplo si el usuario al utilizar algún navegador se va de página en página, de tal suerte que llega a una y navegando, regresa a la misma donde estaba el ***applet,*** entonces se llama otra vez al método ***init.***

stop

Se llama cada vez que el navegador quita el foco del ***applet,*** o bien termina sus operaciones, porque el navegador guiado por el usuario se mueve a otra página.

destroy

Es llamado cada vez que el ***applet,*** terminó de utilizarse y va a ser eliminado de la memoria.

paint

Se llama cuando se dibuja nuevamente el ***applet,*** pasando un objeto de la clase ***Graphics*** y usando en los métodos del objeto a dibujaren el ***applet.***

update

Sirve para cuando se va a volver a dibujar una parte del ***applet.***

V.3.- Restricciones de seguridad

Como se mencionó al principio de este capitulo, los ***applets*** tienen sus ventajas y desventajas, entre las que se encuentran precisamente las referentes a las de seguridad y son las que a continuación se comentan:

- Protegen la integridad de los datos y de la información, ya que no puede tocar ningún disco duro o local.
- No puede ejecutar comandos del sistema operativo, con esto evitan que por equivocación o por mala fe como algún cracker o hacker puedan afectar el funcionamiento del sistema.

- Prevé errores de inconsistencia, por realizar una actualización instantánea, debido a que la actualización del ***applet*** es del lado del servidor y se ve reflejada cualquier actualización del lado del cliente.

V.4.- Paso de parámetros

Cuando se crea la aplicación de HTML (también se pueden introducir etiquetas <APPLET> en el código del arhivo con extensión java), para crear una página Web se usa la etiqueta <APPLET>, para visualizar el ***applet,*** donde se ponen varios parámetros entre los que se describen a continuación algunos de los más utilizados, de acuerdo a la siguiente sintaxis:

```
<APPLET
        [CODEBASE =URL]
        CODE = nombre del archivo
        [ALT = texto alterno]
        [NAME = nombre de instancia]
        WIDTH = pixeles
        HEIGHT = pixeles
        [ALIGN = alineación]
        [VSPACE = pixeles]
        [HSPACE = pixeles]
>
[ <PARAM NAME = nombre VALUE = valor>]
.
.
.
.
[ <PARAM NAME = nombre VALUE = valor>]
</APPLET>
```

Donde:

- CODEBASE: URL dice el directorio en donde se busca el código del ***applet.***
- CODE, es el nombren del archivo del ***applet,*** con todo y extensión ***.class.***
- ALT, es el texto de error, que se va a desplegar, si es que el navegador soporta el ***applet,*** pero no lo puede ejecutar por alguna causa.
- NAME, es el nombre del ***applet*** en el navegador, ya que es necesaio darles nombres a los ***applets*** para poder buscar otras ***applets.***
- WIDTH y HEIGHT, es la anchura y la altura respectivamente, del espacio reservado para el ***applet.***
- ALIGN, pudiendo ser LEFT, RIGHT, TOP, BOOTOM, MIDDLE, etc, para especificar la alineación del ***applet.***
- VSPACE, es el espacio ubicado sobre y desde el ***applet***
- HSAPCE, Es el espacio que se ubica a la derecha e izquierda del ***applet***
- PARAM NAME, es el nombre del parámetro que se le va a pasar al ***applet.***
- PARAM VALUE, es en sí el valor del parámetro.

NOTA: la convención que se siguió es que lo que se encuentra entre los paréntesis cuadrados [y], son comandos opcionales a diferencia de los ejemplos anteriores, que se ponían con pico paréntesis, porque se podían confundir con los comandos de HTML.

VI: PROGRAMACIÓN CLIENTE/SERVIDOR

VI.1.- Arquitectura de las aplicaciones para red

Por lo general las aplicaciones Web, son procesadas, por entero del lado del servidor, trayendo consigo un uso excesivo de memoria, manteniendo al usuario en espera mientras termina de ejecutarse dicha aplicación. Pero los navegadores que agrega JAVA, del lado del cliente (usuario), pueden ejecutar aplicaciones, aparte de desplegar los documentos HTML, poniendo a correr el proceso en el lugar adecuado.

VI.2- Implantación de clientes y servidores

VI.3.- Interfaz JDBC

Introducción a JDBC

Para poder entender ***JDBC***, es necesario recordar un poco sobre el lenguaje de consulta ***SQL.*** El cual es un lenguaje para crear, manejar y examinar las tablas de las bases de datos relacionales, es un lenguaje de cuarta generación (4GL), de aplicación específica, que con comandos pequeños, llega a realizar operaciones de alto nivel, como lo son las consulta, generación de reporte, ordenamientos, etc.

SQL se estandarizo en el año de 1992, con el fin de que cualquier programa pudiera comunicarse, con la mayoría de los sistemas de las bases de datos, sin cambiar los comandos ***SQL,*** sin embargo cada base de datos, cuenta con una interfase diferente, así como diferentes extensiones de ***SQL,*** proporcionada por cada proveedor, por lo que fue necesario crear un interfase basada en **C** llamada **ODBC,** para estandarizar la comunicación con una base de datos, basada en ***SQL.***

SQL, es un lenguaje muy poderoso, pero esta enfocado solo al manejo de las bases de datos, para realizar aplicaciones en otras áreas, como cálculos, despliegues gráficos, etc., se deben de ocupar otros lenguajes de programación de aplicaciones generales, como los que han tenido auge en los tiempos más recientes como los orientados a objetos (POO), como pediera ser C++, pero la desventaja que tienen es que todavía vienen arrastrando la gran dependencia a una sola plataforma, siendo JAVA, tal vez el primero que logra esta independencia, gracias a la utilización de su código intermedio *bytecode*.

JAVA maneja librerías estándar, entre las que se encuentra JDBC, la cual es una evolución de ODBC y la compañía que creo JAVA (JavaSoft), realizó un puente JDBC-ODBC, para lograr una portabilidad en el uso de las bases de datos.

Para poder utilizar una aplicación en JAVA que utilice JDBC, es necesario contar con las siguientes acciones:

- Crear una base de datos.- A través de herramientas distintas a las de JAVA y que soporten el manejo por medio de ***SQL.*** Y cargar el puente JDBC-ODBC. Para lo cual se requiere la clase JdbcOdbcDriver, que tiene el formato:

ClassForName(“sun.jdbc.odbc.JdbcOdbcDriver”);

- Realizar la conexión a un ODBC Data Source.- La cual es una base de datos, que cuenta con el drive de ODBC.
  Por ejemplo
  Connection con = DriverManager.getConnection(url,”login”,”password”);
- Introducir los datos/información a la base de datos.- Por medio de sentencias como las que se muestran a continuación:

  Statement stmtcon.createStatement( );

- Consultar la información de la base de datos. Para lo cual se ocupa el método de executeQuery con que cuenta JAVA y los resultados son devueltos en un objeto tipo ResultSet y pueden ser analizados renglón por renglón, a través de los métodos ResultSetNext y ResuktSet.getXXX

VI.4.- Implantación de servicios en línea

ANEXO A

PROGRAMA ORIGINAL DE LA ASIGNATURA: PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA

	CLAVE: 2040
PLAN:98	CRÉDITOS: 8
LICENCIATURA: **INFORMÁTICA**	SEMESTRE : 9º
AREA: INFORMÁTICA	HRS. CLASE: 2
REQUISITOS: NINGUNO	HRS. POR SEMANA: 4

TIPO DE ASIGNATURA: OBLIGATORIA ( ) OPTATIVA (x)

OBJETIVO GENERAL: PRESENTAR AL PARTICIPANTE LOS CONCEPTOS FUNDAMENTALES DE

LA PROGRAMACIÓN ORIENTADA A OBJETOS APLICÁNDOLOS CON EL USO DEL LENGUAJE JAVA PARA EL DESARROLLO DE APLICACIONES GENERALES	
TEMAS:	HORAS SUGERIDAS:
I - INTRODUCCIÓN	6
II. CONCEPTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA	16
III. ESTRUCTURA DEL LENGUAJE	16
IV. PROGRAMACIÓN DE INTERFASES GRÁFICAS	8
V. PROGRAMACIÓN DE APPLETS	8
VI: PROGRAMACIÓN CLIENTE/SERVIDOR	8
EVALUACIÓN	6
TOTAL	68

TEMAS:
I- INTRODUCCIÓN **I.1- El paradigma orientado a objetos** **I.2- Principios fundamentales de la POO** **I.3.- Análisis y diseño orientado a objetos** **I.4.- Lenguajes orientados a objetos** **II. CONCEPTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA** **II.1.- Clases de Objetos**

II.2.- Métodos y atributos de los objetos
II.3.- Herencia
II.4- Polimorfismo

III. ESTRUCTURA DEL LENGUAJE
III.1.- Tipos de datos
III.2.- Estructuras de control
III.3.- Paquetes de JAVA
III.4.- Programación con excepciones
III.5.- Cláusula try catch

IV. PROGRAMACIÓN DE INTERFASES GRÁFICAS
IV.1.- El paquete JAVA.awt
IV.2.- Componentes y elementos gráficos
IV.3.- Modelo para el manejo de eventos

V. PROGRAMACIÓN DE APPLETS
V.1.- Declaración de un applet
V.2.- Ciclo de vida
V.3.- Restricciones de seguridad
V.4.- Paso de parámetros

VI: PROGRAMACIÓN CLIENTE/SERVIDOR
VI.1.- Arquitectura de las aplicaciones para red
VI.2- Implantación de clientes y servidores
VI.3.- Interfaz JDBC
VI.4.- Implantación de servicios en línea

ANEXO B

PROGRAMA DE ***PROPUESTO*** DE LA ASIGNATURA: PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA

	CLAVE: 2040
PLAN:98	CRÉDITOS: 8
LICENCIATURA: **INFORMÁTICA**	SEMESTRE : 9º
AREA: INFORMÁTICA	HRS. CLASE: 2
REQUISITOS: : PROGRAMACIÓN DE LENGUAJES DE CUARTA GENERACIÓN	HRS. POR SEMANA: 4

TIPO DE ASIGNATURA: OBLIGATORIA ( ) OPTATIVA (x)

OBJETIVO GENERAL: PRESENTAR AL PARTICIPANTE LOS CONCEPTOS FUNDAMENTALES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS APLICÁNDOLOS CON EL USO DEL LENGUAJE JAVA PARA EL DESARROLLO DE APLICACIONES GENERALES	
TEMAS:	HORAS SUGERIDAS:
I – INTRODUCCIÓN	10
II. ESTRUCTURA DEL LENGUAJE	16
III. CONCEPTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA	16
IV. PROGRAMACIÓN DE INTERFASES GRÁFICAS	10
V. PROGRAMACIÓN DE APPLETS	10
EVALUACIÓN	6
TOTAL	68

TEMAS:
Objetivo: Presentar un panorama general del paradigma de la Programación Orientada a Objetos, así como los conceptos básicos que se requieren para comprender este tipo de programación. **I- INTRODUCCIÓN** **I.1- El paradigma orientado a objetos** **I.2- Principios fundamentales de la POO**

I.3.- Análisis y diseño orientado a objetos
I.4.- Lenguajes orientados a objetos

Objetivo: Dar a conocer los elementos principales de que consta la sintaxis y la semántica del lenguaje de programación JAVA.

II. ESTRUCTURA DEL LENGUAJE
II.1.- Tipos de datos
II.2.- Estructuras de control, constructores, creación de objetos y recolección de basura
II.3.- Paquetes de JAVA
II.4.- Programación con excepciones
II.5.- Cláusula try catch

Objetivo: Mostrar brevemente el funcionamiento de la plataforma empleada (la Java Virtual Machine JVM), así como el proceso de creación, compilación y ejecución de una aplicación hecha en JAVA.

III. CONCEPTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA
III.1.- Introducción a JAVA
III.2.- Cómo instalarlo
III.2.- Clases de Objetos
III.3.- Constructores y Destructores
III.4.- Métodos
III.5.- Herencia
III.6- Polimorfismo

Objetivo: Proporcionar los elementos para la utilización de los paquetes básicos de JAVA, para poder utilizar un ambiente gráfico.

IV. PROGRAMACIÓN DE INTERFASES GRÁFICAS
IV.1.- El paquete JAVA.awt
IV.2.- Componentes y elementos gráficos
IV.3.- Modelo para el manejo de eventos

Objetivo: Dar los conocimientos básico, para poder elaborar y ejecutar programas de tipo Applet.

V. PROGRAMACIÓN DE APPLETS
V.1.- Declaración de un applet
V.2.- Ciclo de vida
V.3.- Restricciones de seguridad
V.4.- Paso de parámetros

BIBLIOGRAFÍA BÁSICA

1. **DEITEL H.M. Y DEITEL P.J.**, *Como programar en C/C++*, (2ª edición.), MÉXICO, PEARSON EDUCACIÓN PRENTICE-HALL,1995, 927 pp.

2. **EUÁN AVILA JORGE IVAN Y CORDERO BORBOA LUIS GONZAGA.**, *Estructuras de datos*, (1ª reimpresión.), MÉXICO, LIMUSA, tomada de la primera edición de la UNAM (FACULTAD DE INGENIERÍA), 1989, 219 pp.

3. **GOODRICH MICHAEL T. Y TAMASSIA ROBERTO** , *Estructuras de Datos en Java*, (2ª. Edición en inglés y 1ª en español.), MÉXICO, COMPAÑÍA EDITORIAL CONTINENTAL GRUPO PATRIA CULTURAL, 2002, 641 pp.

4. **LANGSAM YEDIDYAH, AUGENSTEIN MOSHE J. Y TENENBAUM AARÓN M.** , *Estructuras de Datos con **C** y **C**++*, (2ª. edición.), MÉXICO, PRENTICE-HALL HISPANOAMERICANA, 1997, 672 pp.

5. **RUMBAUGH JAMES, BLAH MICHAEL, PREMERLANI WILLIAMS, HEDÍ FREDERICK Y LORENSEN WILLIMAS.** *Modelado y diseño orientado a objetos METODOLOGÍA OMT (*S/E), ESPAÑA, PRENTICE-HALL, España 1996, págs. 643.

BIBLIOGRAFÍA COMPLEMENTARIA

1. CEBALLOS, Francisco J., *Enciclopedia de Microsoft Visual Basic 6,* España, Alfa omega-Rama, 2002.
2. CEBALLOS, Francisco J., *Microsoft Visual Basic 6. curso de programación,* España, Alfa omega-Rama, 2002.

SUGERENCIAS DIDÁCTICAS:

Exposición audiovisual	(X)
Exposición oral	(X)
Ejercicios dentro de la clase	(X)
Seminarios	( )
Lecturas obligatorias	(X)
Trabajos de investigación	(X)
Prácticas de taller o laboratorio	(X)
Prácticas de campo	( )
Otras	( )

SUGERENCIAS PARA LA EVALUACIÓN:

Exámenes parciales	(X)
Exámenes finales	(X)
Trabajos y tareas fuera de aula	(X)
Participación en clase	(X)
Asistencia a prácticas	( )
Otras	( )

Es de hacer notar, que podría tenerse una segunda asignatura, que fuera la continuación de esta. o bien que se tuviera como antecedente el manejo de bases de datos y la programación en C/C++.